ДРОГОБИЦЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Кафедра фізики та інформаційних систем

Ольга ГАРБИЧ-МОШОРА, Христина ВОЙТОВИЧ

Операційні системи

122 «Комп'ютерні науки»

Дрогобич – 2024

УДК 004.451(072)
Г20

Рекомендовано вченою радою Дрогобицького державного педагогічного університету імені Івана Франка (протокол № 12 від 29.11.2024 р.)

Рецензенти:

Андрій ГРИГОРОВИЧ, кандидат технічних наук, вчитель інформатики і фізики КЗ ЛОР "Обласний науковий ліцей";

Оксана СІКОРА, кандидат технічних наук, доцент кафедри фізики та інформаційних систем Дрогобицького державного педагогічного університету імені Івана Франка.

Відповідальний за випуск –

Роман ЛЕШКО, кандидат фізико-математичних наук, доцент кафедри фізики та інформаційних систем Дрогобицького державного педагогічного університету імені Івана Франка.

Гарбич-Мошора Ольга, Войтович Христина

Операційні системи: навчально-методичний посібник для студентів спеціальності 122 «Комп'ютерні науки». Дрогобич : Дрогобицький державний педагогічний університет імені Івана Франка, 2024. 84 с.

У навчально-методичному посібнику запропоновані методичні вказівки до виконання лабораторних робіт з дисципліни «Операційні системи», спрямовані на формування навичок та вмінь у галузі розробки та управління системними ресурсами. Мета посібника – ознайомити студентів із сучасними методами взаємодії з операційними системами, а також надати їм систематизовану інформацію та практичні рекомендації для успішного виконання лабораторних робіт з цієї дисципліни.

Навчально-методичний посібник розроблено відповідно до програми навчальної дисципліни «Операційні системи» для підготовки фахівців напряму підготовки 12 Інформаційні технології, спеціальності 122 Комп’ютерні науки, затвердженої вченою радою Дрогобицького державного педагогічного університету імені Івана Франка.

У ньому розглядаються ключові аспекти роботи з операційними системами, зокрема функціонування системних компонентів, управління процесами, пам’яттю, файловими системами та основами програмування в середовищі Windows API. Кожна лабораторна робота містить теоретичні відомості, практичні завдання та рекомендації щодо їх виконання, що допоможе студентам краще зрозуміти принципи роботи операційних систем і набути практичних навичок, необхідних для професійної діяльності у сфері інформаційних технологій.

ЗМІСТ

ВСТУП

Використання системних можливостей операційної системи є важливим елементом для розробників програмного забезпечення, що дає їм змогу безпосередньо взаємодіяти з ключовими компонентами операційної системи та апаратного забезпечення. Сучасні операційні системи, такі як Windows, надають багатий набір функцій та інтерфейсів, які надають програмістам можливість ефективно управляти ресурсами системи, такими як пам’ять, процеси, файлові системи й елементи інтерфейсу користувача.

Цей навчально-методичний посібник охоплює кілька ключових аспектів програмування у середовищі Windows API, що надає доступ до системних ресурсів і можливостей операційної системи Windows. Приклади, пов'язані з операціями з системним часом, реалізацією будильника та таймера зворотного відліку, демонструють використання WinAPI для виконання низькорівневих функцій, які є фундаментальними для ефективного програмування у середовищі Windows.

Навчання маніпуляцій із такими системними елементами не тільки розширює знання студентів про роботу комп'ютерних систем, але й допомагає створювати більш комплексні та інтерактивні програми, що можуть виконувати завдання автоматизації і моніторингу процесів. Вивчення основ операційних систем, їх архітектури та принципів роботи дає студентам розуміння, як забезпечується взаємодія програмного забезпечення з апаратними ресурсами. Знання цих концепцій є критично важливими для розробників, оскільки вони впливають на ефективність, продуктивність і надійність створюваних програм.

Отже, можемо стверджувати, що пропонований навчально-методичний посібник дає змогу студентам не тільки отримати знання про використання системних функцій Windows, а й набути практичних навичок, необхідних для розробки професійних програмних продуктів. Його метою є ознайомлення студентів із основами роботи з системними функціями Windows API, а також закріплення цих навичок через виконання практичних завдань.

У рамках посібника будуть розглянуті ключові концепції, пов’язані з управлінням системним часом, а також реалізацією програмних рішень для практичного використання. Окрім базових функцій Windows API, студентам буде запропоновано вивчити принципи проєктування інтерфейсу користувача, що робить програми зручними й інтуїтивно зрозумілими.

Цей посібник стане важливим інструментом для формування практичних навичок програмування на C++, а також для розвитку аналітичного мислення та здатності до самостійного вирішення завдань. Зосередження на реальних прикладах і задачах допоможе студентам краще засвоїти матеріал, а також підготуватися до подальшого навчання у галузі розробки програмного забезпечення.

У підсумку, методичні рекомендації стануть надійною основою для студентів, які прагнуть поглибити свої знання та практичні навички у сфері програмування, а також для тих, хто планує розвиватися в області створення програмного забезпечення для платформ Windows. Виконання завдань, описаних у посібнику, не лише надасть корисні знання, але й стимулюватиме інтерес до подальшого вивчення інформаційних технологій.

Для виконання лабораторних робіт ми використовуємо **Visual Studio Code (VS Code)**. Основні переваги використання VS Code:

- **Зручний термінал**: можливість працювати з інтегрованим терміналом без необхідності перемикатися між вікнами.
- **Розширення для мов програмування**: легке встановлення та використання плагінів для різних мов програмування.
- **Синтаксичне підсвічування**: допомагає краще читати та писати скрипти або програми.
- **Можливість роботи в різних ОС**: підтримка як Linux, так і Windows, робить VS Code універсальним інструментом для роботи з різними операційними системами.

Linux:

1. Відкриття терміналу у VS Code:

- Натисніть `Ctrl + `` для відкриття інтегрованого терміналу прямо у VS Code.
- Використовуйте команди, такі як top, htop, ps, або systemctl у терміналі VS Code.

2. Налаштування для Python або C/C++:

- Встановіть відповідні розширення для програмування, наприклад Python або C/C++.
- Ви можете писати та запускати програми одночасно у VS Code і спостерігати за процесами через термінал.

Windows:

1. PowerShell або командний рядок у VS Code:

- Відкрийте PowerShell або командний рядок через інтегрований термінал (`Ctrl + ``).
- Використовуйте команди для роботи з процесами і файловими системами, наприклад, Get-Process.

2. Налаштування для PowerShell:

- Встановіть розширення для PowerShell у VS Code.
- Використовуйте PowerShell-скрипти для моніторингу процесів, планувальника та інших компонентів системи.

Лабораторна робота № 1

ВИВЧЕННЯ АРХІТЕКТУРИ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX/WINDOWS

Мета:

Ознайомитися з основними компонентами операційної системи (ядро, планувальник, файлові системи, підсистеми входу / виводу), а також дослідити систему планування завдань в операційній системі.

ТЕОРЕТИЧНІ ВІДОМОСТІ

1. Ознайомлення з компонентами операційної системи (C++).

Для виконання цього завдання створюємо програму на C++, яка буде взаємодіяти з операційною системою для отримання інформації про компоненти. Програма буде залежати від операційної системи, інші ОС мають свої механізми роботи з процесами та ресурсами.

Linux:

1. *Перевірка інформації про ядро*. Для отримання інформації про версію ядра використовуємо команду *uname* через системний виклик *system()*.

```cpp
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Kernel version: " << std::endl;
    system("uname -r");
    return 0;
}
```

2. *Аналіз файлової системи.* Для перегляду типів файлових систем можна використовувати команду *df*. Ось як це можна зробити:

```
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Типи файлових систем: " << std::endl;
    system("df -T");
    return 0;
}
```

3. *Моніторинг системних журналів (вводу / виводу.* Щоб переглянути системні логи через C++, використовуємо команду *dmesg:*

```
#include <iostream>
#include <cstdlib>

int main() {
```

Windows:

1. *Перевірка інформації про ядро***.** Отримання версії ядра Windows через PowerShell:

```
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Версія ядра Windows: " << std::endl;
    system("powershell systeminfo | findstr /B /C:\"OS Version\"");
    return 0;
}
```

2. *Перегляд файлових систем*. Збереження інформації про активні диски за допомогою PowerShell:

```cpp
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Активні диски: " << std::endl;
    system("powershell fsutil fsinfo drives");
    return 0;
}
```

*Результат виконання***:**

Коли ви запустите цей код у Windows, він виведе щось подібне до:

```
Інформація про версію операційної системи Windows:

OS Version:                10.0.19044 N/A Build 19044
```

Примітка: Вивід буде варіюватися залежно від версії Windows, встановленої на вашій системі. Якщо система має іншу версію (наприклад, Windows 11), відповідно вивід змінитися.

Цей рядок є виводом команди *systeminfo*, яка показує операційну версію системи Windows. Ось детальний аналіз інформації, яку ви отримали:

- **Версія ОС**: Це позначає операційну версію системи.
- **10.0.19044**:
    - **10.0**: основний номер версії Windows. Номер 10 вказує, що ви використовуєте Windows 10.
    - **19044**: номер оновлення для цієї версії. У вашому випадку це оновлення, яке додає нові функції або виправлення безпеки.

- **N/A**: традиційно вказує на те, що інформація про конкретну частину системи (у цьому випадку, можливо, про редакцію) недоступна.
- **Збірка 19044**: номер складання (номер збірки) Windows. Номери складання програми для ідентифікації конкретних збірок Windows, які можуть відновити зміни чи виправлення.

2. Моніторинг процесів і ресурсів

Linux:

1. *Моніторинг процесів за допомогою утиліти* ***top.*** Використовуйте команду для виведення активних процесів:

```
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Активні процеси (top): " << std::endl;
    system("top");
    return 0;
}
```

2. *Моніторинг процесів через* ***htop*** *(зручний інтерфейс)* Якщо *htop* не встановлено, встановіть:

```
#include <iostream>
#include <cstdlib>

int main() {
    system("sudo apt install htop");
    std::cout << "Активні процеси (htop): " << std::endl;
    system("htop");
    return 0;
}
```

3. *Моніторинг служби через* ***systemctl.*** Виведення всіх активних служб:

```cpp
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Активні служби: " << std::endl;
    system("systemctl list-units --type=service");
    return 0;
}
```

Windows:

1. *Моніторинг процесів через диспетчер завдань*. Використовуйте диспетчер завдань для перегляду поточних процесів.

2. *Перегляд процесів через PowerShell* Використовуйте команди для виведення списку всіх активних процесів:

```cpp
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Активні процеси: " << std::endl;
    system("powershell Get-Process");
    return 0;
}
```

3. *Аналіз системи планування*

Linux:

Для Linux використовують системні команди для аналізу процесів і планування їх політики. Ми можемо інтегрувати їх через системний виклик у C++.

1. Перегляд політики планування процесів за допомогою команди ***chrt.*** Програма на C++, яка отримує політику планування для конкретного процесу:

```cpp
#include <iostream>
#include <cstdlib>

int main() {
    int pid;
    std::cout << "Введіть PID процесу для перегляду політики планування: ";
    std::cin >> pid;

    std::string command = "chrt -p " + std::to_string(pid);
    std::cout << "Політика планування для процесу з PID " << pid << ":" << std::endl;
    // Виконує команду chrt для відображення політики планування процесу
    system(command.c_str());

    return 0;
}
```

Ця програма запитує у користувача процес *PID* і виконує команду *chrt -p <PID>* для перегляду його політики планування.

2. Аналіз часу процесора для кожного процесу. Застосовуючи *top* або *htop*, ми можемо відстежувати використання процесорного часу кожним процесом.

Windows:

Для Windows ми можемо використовувати PowerShell або диспетчер завдань для аналізу процесів та їх пріоритетів.

1. Перегляд пріоритету процесів за допомогою PowerShell. Програма на C++ для отримання інформації про важливі процеси:

```cpp
#include <iostream>
#include <cstdlib>

int main() {
    std::cout << "Перегляд інформації про пріоритет процесів у Windows..." << std::endl
    // Виконує команду PowerShell для отримання інформації про пріоритети процесів
    system("powershell -Command \"Get-Process | Select-Object Id, PriorityClass\"");

    return 0;
}
```

Ця програма виконує команду PowerShell, яка виводить список процесів та їх пріоритети.

2. Установка пріоритету для процесів вручну. Програму для встановлення параметру вручну можна створити за допомогою API Windows, але для базового керування користувач може відкрити диспетчер завдань і встановити параметри вручну.

ЗАВДАННЯ

1. Порівняння версій ядра для Linux і Windows. Запишіть отриманий результат. Чим відрізняються ці версії? Яка інформація більш детальна?

2. Порівняйте типи файлових систем, які використовують у Linux та Windows. Які типи є найбільш розширеними для кожної ОС? У чому полягають ключові відмінності?

3. Зробіть список активних процесів і їх використання ресурсів на кожній з ОС. Проаналізуйте загальну кількість процесів та навантаження на систему.

4. Проаналізуйте, як обидві ОС керують ресурсами процесора і пам'яті. Наскільки ефективна система планування обох ОС справляється з навантаженням? Які процеси мають вищу перевагу і як це впливає на загальну продуктивність?

4. До звіту підтвердження виконаних кроків додайте покрокові скріншоти.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яку основну мету має ця лабораторна робота?
2. Як перевірити версію ядра операційної системи Linux за допомогою терміналу?

3. Яка команда використовує для отримання інформації операційну систему Windows?
4. Які інструменти використовують для моніторингу системних ресурсів у Linux?
5. Яка команда дає змогу переглядати активні процеси в Linux?
6. Як можна вручну встановити пріоритетний процес в операційній системі Windows?
7. Що таке планувальник процесів в операційній системі?
8. Як за допомогою команди `systemctl` переглянути активні служби в Linux?
9. Яка команда використовує для завершення процесу в Windows?
10. Що відображає команда `df -T` в Linux?
11. Як ви можете переглянути список усіх активних процесів у Windows за допомогою PowerShell?
12. Як відкрити інтегрований термінал у Visual Studio Code?
13. Як за допомогою C++ відкрити системні команди в програмі?
14. Яке розширення для Visual Studio Code варто встановити для роботи з C++?
15. Чому важливо використовувати моніторинг процесів під час роботи з операційними системами?

Лабораторна робота № 2

WIN API. ПРОЦЕСИ В ОПЕРАЦІЙНІЙ СИСТЕМІ

Мета:

- Ознайомитися з основними методами керування процесами в операційній системі.
- Розробити консольну програму на C++ для керування процесами в ОС Windows

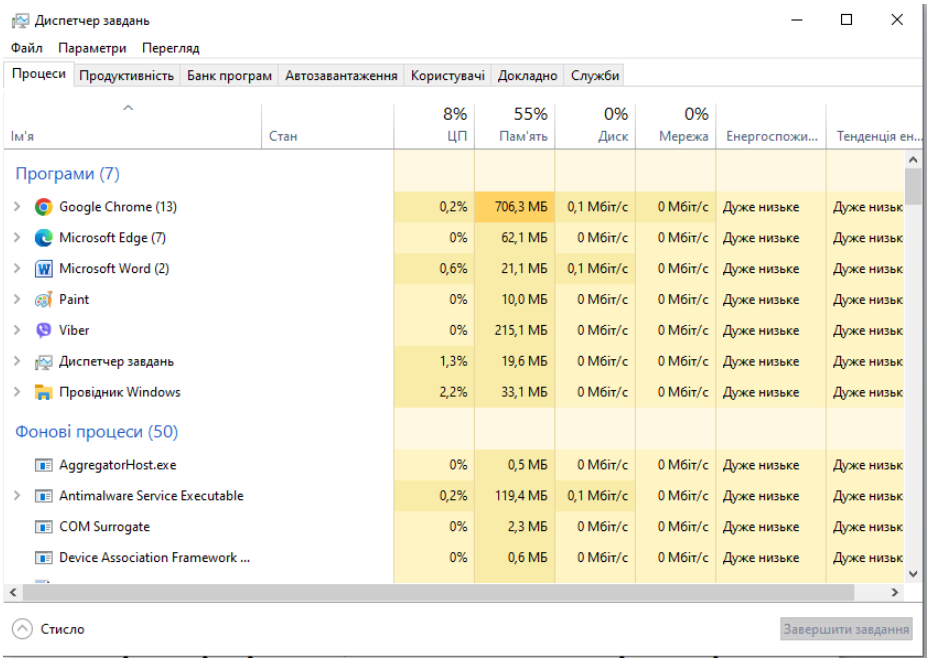
ТЕОРЕТИЧНІ ВІДОМОСТІ

Процеси в операційній системі **Windows** – це виконувані екземпляри програм, які виконуються в оперативній пам'яті комп'ютера. Кожен процес має власний адресний простір пам'яті і виконує конкретні функції у системі. Кожен процес **Windows** отримує унікальний ідентифікатор процесу **(Process ID - PID)** і може незалежно завантажуватися і розвантажуватися операційною системою (у тому числі програмно).

У вікні *WindowsTask Manager (Диспетчер завдань)* є вкладка *Processes (Процеси)*, на якій можна переглядати різні статичні дані про процеси, що виконуються на цій машині, з-поміж іншого і їхні **PID**-ідентифікатори та імена образів.

При запуску програм операційна система створює для нього окремий процес, якому надається певний адресний простір у пам'яті, ізольований від інших процесів. Процес може мати кілька потоків, однак, як мінімум, містить один – головний.

У додатку на C++, точкою входу в програму є метод **Main**. Виклик цього методу автоматично створює головний потік, з якого уже можуть запускатися вторинні.



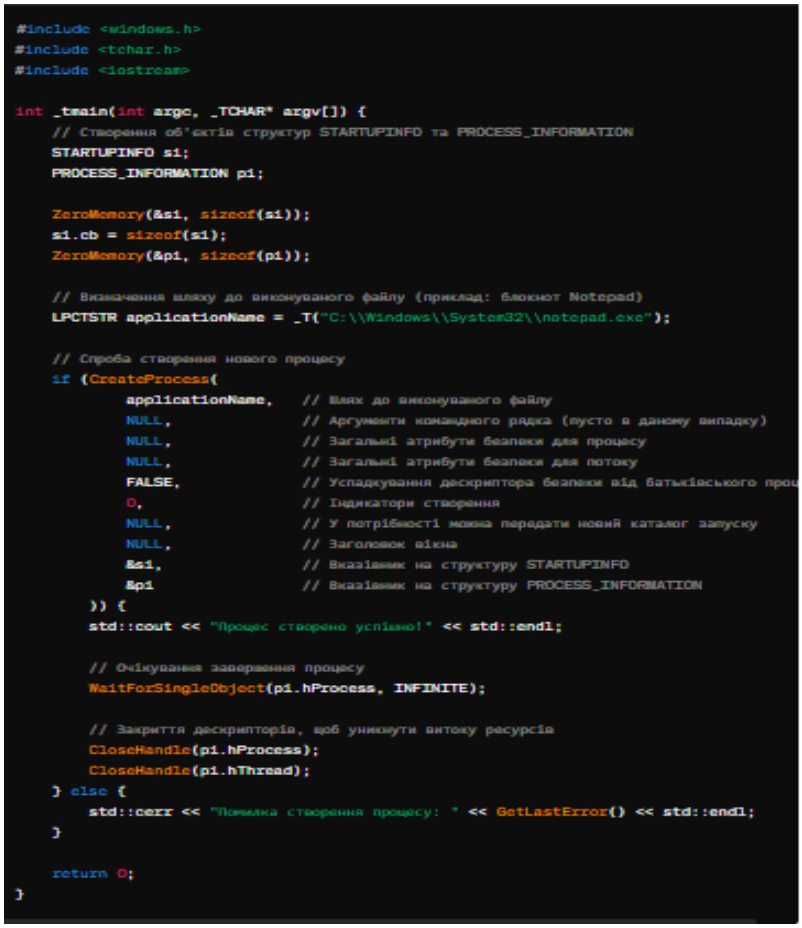
У **.NET Framework** клас **Process** розміщується у просторі імен **System.Diagnostics**, уможливлюючи управління процесами в операційній системі.

Клас **Process** пропонує низку методів та властивостей для виконання таких операцій, як:

Створення нового процесу. У мові **C++** можна використовувати різні методи для створення нового процесу. Один зі способів – використання функцій, які забезпечують створення нового процесу, таких як **fork** для **Unix**-подібних систем або **CreateProcess** для **Windows.** Однак нативної функції **Process.Start,** як у **.NET Framework**, у мові **C++** не існує.

Для запуску нового процесу на **C++** ви можете використовувати функції з **Windows API** або сторонні бібліотеки. Нижче наведено приклад використання функцій з **Windows API** для створення нового процесу (відкриття блокнота).

Отримання інформації про наявний процес. Ви можете використовувати властивості і методи для отримання інформації про наявний процес. У мові **C++** немає вбудованого класу, аналогічного **System.Diagnostics.Process** у **.NET Framework,** який надає властивості та методи для отримання інформації про такий процес. Однак існують системні виклики та бібліотеки, які можна використовувати для отримання інформації про процеси в операційній системі.



```cpp
#include <windows.h>
#include <tchar.h>
#include <iostream>

int _tmain(int argc, _TCHAR* argv[]) {
    // Створення об'єктів структур STARTUPINFO та PROCESS_INFORMATION
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory(&pi, sizeof(pi));

    // Визначення шляху до виконуваного файлу (приклад: блокнот Notepad)
    LPCTSTR applicationName = _T("C:\\Windows\\System32\\notepad.exe");

    // Спроба створення нового процесу
    if (CreateProcess(
            applicationName,   // Шлях до виконуваного файлу
            NULL,              // Аргументи командного рядка (пусто в даному випадку)
            NULL,              // Загальні атрибути безпеки для процесу
            NULL,              // Загальні атрибути безпеки для потоку
            FALSE,             // Успадкування дескриптора безпеки від батьківського проц
            0,                 // Індикатори створення
            NULL,              // У потрібності можна передати новий каталог запуску
            NULL,              // Заголовок вікна
            &si,               // Вказівник на структуру STARTUPINFO
            &pi                // Вказівник на структуру PROCESS_INFORMATION
        )) {
        std::cout << "Процес створено успішно!" << std::endl;

        // Очікування завершення процесу
        WaitForSingleObject(pi.hProcess, INFINITE);

        // Закриття дескрипторів, щоб уникнути витоку ресурсів
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
    } else {
        std::cerr << "Помилка створення процесу: " << GetLastError() << std::endl;
    }

    return 0;
}
```

Ви повинні замінити значення **targetPID** на ідентифікатор процесу, для якого хочете отримати інформацію. У цьому прикладі ми використовуємо шлях **/proc/{PID}/status**, який містить різноманітну інформацію про процес.

Важливо відзначити, що цей підхід специфічний для **Unix**-подібних систем, і в **Windows** інші методи використовуються для отримання інформації про процеси.

*Управління наявним процесом***.** Для управління процесами в операційній системі **Windows** можна використовувати функцію **TerminateProcess,** яка дає змогу завершувати поточний процес за його ідентифікатором процесу **(PID).** Ось приклад коду на мові C++ для використання цієї функції.

Важливо відзначити, що використання **TerminateProcess** може призвести до некоректного завершення процесу, і в результаті – порушити його ресурси, тому цим необхідно користуватися з обережністю. Якщо є можливість, то для комунікації з процесами та їх контрольованого завершення краще використовувати інші методи.

Давайте розглянемо низку властивостей класу **Process в .NET** у контексті C++:

- ✓ **Handle**: повертає дескриптор процесу.
- ✓ **Id**: отримує унікальний ідентифікатор процесу в рамках поточного сеансу ОС.
- ✓ **MachineName:** повертає ім'я комп'ютера, на якому запущений процес.
- ✓ **Modules:** отримує доступ до колекції ProcessModuleCollection, яка зберігає набір модулів (файлів dll і exe), завантажених у рамках цього процесу.
- ✓ **ProcessName:** повертає ім'я процесу, яке нерідко збігається з ім'ям програми.
- ✓ **StartTime**: повертає час, коли процес був запущений.
- ✓ **VirtualMemorySize64**: повертає обсяг пам'яті, виділений для цього процесу
- ✓ **BasePriority:** отримує базовий пріоритет процесу.
- ✓ **EnableRaisingEvents:** вказує, чи включено спрацьовування подій для процесу.
- ✓ **ExitCode:** отримує код завершення процесу.
- ✓ **ExitTime:** отримує час завершення процесу.
- ✓ **HasExited:** вказує, чи процес вже завершено.
- ✓ **MainWindowHandle:** отримує дескриптор вікна головного вікна процесу (якщо таке вікно є).

Давайте розглянемо деякі методи класу **Process в .NET** C++:

- ✓ **Метод CloseMainWindow ():** закриває вікно процесу, який має графічний інтерфейс
- ✓ **Метод GetProcesses ():** повертає масив усіх запущених процесів
- ✓ **Метод GetProcessesByName ():** повертає процеси по його імені. Оскільки можназапустити кілька копій однієї програми, то повертає масив.
- ✓ **Метод Kill ():** зупиняє процес.
- ✓ **Метод Start ():** запускає новий процес.

- ✓ **Refresh():** оновлює інформацію про стан процесу.
- ✓ **WaitForExit():** чекає, доки поточний процес не завершить роботу.
- ✓ **WaitForInputIdle():** Чекає, доки процес не стане неактивним для введення.

Важливо відзначити, що в **C++** для роботи з процесами на різних операційних системах можуть використовуватися різні бібліотеки та системні виклики. Ці властивості та методи дають змогу отримувати додаткову інформацію про процес і взаємодіяти з ним у різних сценаріях.

Приклад простого коду на C++, який використовує клас **Process** для отримання і виведення інформації про всі запущені процеси. Цей код використовує функції **Windows API**, такі як **CreateToolhelp32Snapshot, Process32First і Process32Next**, для отримання та виведення інформації про всі запущені процеси. Будь ласка, зауважте, що цей приклад специфічний для **Windows** і не буде працювати на інших операційних системах.

Для отримання і виведення інформації про всі запущені процеси та знаходження ID процесу, який представляє **Visual Studio**, можна використати аналогічний підхід до використання **Windows API** та функцій **CreateToolhelp32Snapshot, Process32First** і **Process32Next**. Проте важливо враховувати, що **Visual Studio** може мати кілька процесів (наприклад, **devenv.exe**), і вам може знадобитися додаткова логіка для визначення саме потрібного процесу.

Для отримання і виведення інформації про всі запущені процеси та знаходження ID процесу, який представляє **Visual Studio**, можна використати аналогічний підхід до використання **Windows API** та функцій **CreateToolhelp32Snapshot, Process32First** і **Process32Next**. Проте, важливо враховувати, що **Visual Studio** може мати кілька процесів (наприклад, **devenv.exe**), і вам може знадобитися додаткова логіка для визначення саме того процесу, який вам потрібен.

```cpp
#include <iostream>
#include <windows.h>
#include <tlhelp32.h>

int main() {
    // Створення зразка структури PROCESSENTRY32
    PROCESSENTRY32 processEntry;
    processEntry.dwSize = sizeof(PROCESSENTRY32);

    // Отримання дескриптора снапшота процесів
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    if (snapshot == INVALID_HANDLE_VALUE) {
        std::cerr << "Error creating process snapshot." << std::endl;
        return 1;
    }

    // Отримання першого процесу у снапшоті
    if (Process32First(snapshot, &processEntry)) {
        // Виведення інформації про кожен процес
        do {
            std::cout << "Process ID: " << processEntry.th32ProcessID << std::endl;
            std::cout << "Process Name: " << processEntry.szExeFile << std::endl;
            std::cout << "----------------------------------------" << std::endl;
        } while (Process32Next(snapshot, &processEntry));
    } else {
        std::cerr << "Error getting process information." << std::endl;
    }

    // Закриття дескриптора снапшота
    CloseHandle(snapshot);

    return 0;
```

Потоки процесу

Для отримання інформації про потоки процесу в C++**,** ви можете використовувати платформозалежні засоби, такі як функції **Windows API**. Ось кілька з них:

CreateToolhelp32Snapshot: функція створює снапшот даних про процеси або потоки.

Thread32First та Thread32Next: ці функції використовуються для отримання інформації про потоки в снапшоті.

OpenThread: функція відкриває наявний потік за його ідентифікатором.

GetThreadContext: функція отримує контекст виконання потоку.

SuspendThread та ResumeThread: ці функції призначені для зупинки та відновлення виконання потоку.

TerminateThread: функція завершує вказаний потік.

Модулі процесу

Для отримання інформації про модулі (**DLL і EXE** файли), які завантажені у процес, ви можете використати функції з **Windows API в C++.** Ось декілька з них, які вам можуть знадобитися:

EnumProcessModules: функція отримує дескриптори модулів, завантажених в указаний процес.

```
BOOL EnumProcessModules(
  HANDLE  hProcess,
  HMODULE *lphModule,
  DWORD   cb,
  LPDWORD lpcbNeeded
);
```

GetModuleFileNameEx: функція отримує повний шлях до виконуваного файлу модуля.

```
DWORD GetModuleFileNameEx(
  HANDLE  hProcess,
  HMODULE hModule,
  LPWSTR  lpFilename,
  DWORD   nSize
);
```

GetModuleBaseName: функція отримує ім'я модуля (без шляху).

```
DWORD GetModuleBaseName(
  HANDLE  hProcess,
  HMODULE hModule,
  LPWSTR  lpBaseName,
  DWORD   nSize
);
```

Отримання інформації про модулі може бути корисним для аналізу структури і роботи процесу, що відбувається зокрема для визначення використаних бібліотек та ресурсів.

ЗАВДАННЯ

Розробити консольну програму на C++ для керування процесами операційної системи.

Щоб забезпечити унікальність завдань для кожного студента та диференціювати роботу між різними групами, можна поділити їх на три групи, користуючись номерами у журналі. Кожна з них фокусується на одному ключовому елементі, який відрізняється між групами. Усі інші 6 пунктів виконують усі студенти.

Розподіл студентів за групами:

- Група 1 (номери 1–10): фокус на керуванні потоками процесів.
- Група 2 (номери 11–20): фокус на роботі з модулями процесів.
- Група 3 (номери 21–30): фокус на встановленні / зміні пріоритетів процесів.

Програма повинна виконувати такі дії:

1. *Створення процесу*: консольна програма, яка створює новий процес. Вивести інформацію про ідентифікатори процесів та повідомлення про їхній статус.
2. *Вивід усіх процесів*: реалізувати функціонал виведення усіх процесів в режимі реального часу з можливістю автоматичного оновлення.
3. *Завершення вибраного процесу*: розробити можливість завершення вибраного процесу за допомогою введення користувача.
4. *Вивід усіх потоків вибраного процесу*: створити функціонал для виведення усіх потоків вибраного процесу.
5. *Вивід усіх модулів вибраного процесу*: реалізувати можливість виведення усіх модулів вибраного процесу.
6. *Запуск нового процесу*: створити можливість запуску нового процесу із введеними користувачем параметрами, наприклад, відкривання браузера та перехід на вказаний сайт.

Додаткові вимоги для кожної групи

Група 1. Керування потоками

- Реалізувати функцію для **виведення всіх потоків вибраного процесу**. Студенти повинні вивести інформацію про потоки вибраного процесу: їх ID, статус, пріоритет.
- Додати можливість **створення нового потоку** всередині вибраного процесу (якщо можливо) і вивести інформацію про цей потік.

Група 2. Робота з модулями

- Студенти цієї групи повинні реалізувати функцію для **виведення всіх модулів вибраного процесу**, включаючи динамічно підвантажені бібліотеки (DLL).
- Додати можливість **завантаження нового модуля** у вибраний процес (за допомогою системних викликів).

Група 3. Пріоритети процесів

- Реалізувати можливість **встановлення та зміни пріоритету процесу**. Студенти повинні вміти встановлювати різні рівні пріоритету для вибраного процесу (високий, низький, реального часу і т. д.).
- Додати можливість **моніторингу процесу** з урахуванням зміни його пріоритету (як змінилася швидкість виконання або використання ресурсів після зміни пріоритету).

Пояснення:

Кожна група буде мати специфічний фокус, але водночас всі студенти виконують основні завдання з управління процесами. Такий підхід дасть їм можливість заглибитися в конкретну тему і створити унікальніші реалізації.

Створити звіт, в якому детально описано кожне завдання, подано програмний код, пояснені використані концепції та висновки з результатів роботи.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке процес в операційній системі і які його основні характеристики?
2. Як операційна система Windows керує процесами та що таке ідентифікатор процесу (PID)?
3. Яка роль головного потоку в процесі? Як він створюється в програмі на C++?
4. Які методи дають клас Process у .NET Framework для керування процесами?
5. Як створити новий процес в операційній системі Windows за допомогою C++?
6. Які основні функції Windows API використовують для роботи з процесами?
7. Як за допомогою C++ можна отримати інформацію про наявні процеси в системі?
8. Які функції використовуються для управління потоками процесу в C++?
9. Як зупинити процес у Windows за допомогою функції TerminateProcess? Які ризики пов'язані з її використанням?
10. Що таке модулі процесу і як отримати інформацію про них у Windows?
11. Які властивості класу Process отримують інформацію про ресурси процесу, використання пам'яті або час запуску?
12. Які методи класу Process відмовляються від запуску нових процесів або завершення поточних?
13. Що таке знімок процесів, і як він використовується для отримання інформації про потоки та модулі процесів?
14. Як у C++ можна відобразити всі запущені процеси на комп'ютері?
15. Які є засоби контролю та взаємодії з процесами в різних операційних системах (Windows, Unix-подібні системи)?

Лабораторна робота № 3

WIN API. РЕЄСТР ОПЕРАЦІЙНОЇ СИСТЕМИ

Мета:

- Вивчення структури системного реєстру Windows, принципів його функціонування та методів керування ключами реєстру.
- Використання утиліти **regedit** для перегляду та редагування ключів, програмне керування системним реєстром за допомогою класів **Registry** і **RegistryKey** .

ТЕОРЕТИЧНІ ВІДОМОСТІ

Системний реєстр (registry) –ати інформацію про конфігурацію.

Операційна система використовує системний реєстр у такий спосіб:

1. При запуску програми установки Windows (Windows setup), додаванні нового обладнання через панель чи керування запуску програм установки устаткування, операційна система (Configuration manager) додає до реєстру інформацію об устаткування. У реєстрі міститься повний список пристроїв.
2. При установці Windows “поверх” попередньої версії до реєстру переноситься інформація з INI файлів.
3. При додаванні чи видаленні пристроїв, які підтримують специфікацію Plug and Play інформація заноситься в registry.
4. Драйвери пристроїв зчитують з реєстру налаштування, подібно тому, як це робиться при використанні рядка device= файлу CONFIG.SYS.
5. З реєстром працюють різні адміністративні програми, наприклад Панель Керування (Control Panel).

Додавання та видалення ключа реєстру.

Ключовим місцем для зберігання конфігураційних налаштувань в **ОС Windows** є реєстр. Для перегляду реєстру ми можемо скористатися утилітою **regedit.**

Реєстр зберігає набір ключів, у яких є певне значення. Ключі також можуть містити інші ключі.

Усього є п'ять ключів верхнього рівня, призначених для зберігання різної інформації:

- **HKEY_CLASSES_ROOT:** зберігає інформацію про використовування в операційній системі, які програмиспеціальна системна база даних, у якій додатки й операційна система можуть зберіг відкривають той чи той тип файлів, а також інформацію про компоненти COM
- **HKEY_CURRENT_USER**: зберігає лаштування для поточного користувача в системі
- **HKEY_LOCAL_MACHINE**: зберігає інформацію про всі встановлені програми
- **HKEY_USERS:** зберігає налаштування для всіх користувачів
- **HKEY_CURRENT_CONFIG**: зберігає інформацію про конфігурацію встановленого обладнання
- **HKEY_DYN_DATA:** зберігає різні поточні дані
- **HKEY_PERFORMANCE_DATA:** зберігає інформацію про продуктивність додатків

Для управління реєстром у просторі імен **Microsoft.Win32**є два класи: **Registry**і **RegistryKey.**

Registry – це статичний клас, що надає ексклюзивний доступ до ключів реєстру для простих операцій.

Registry – цей клас надає набір стандартних кореневих розділів в реєстрі комп'ютерів, що працюють під управлінням **Windows**. Реєстр є

засобом зберігання відомостей про додатки, користувачів і стандартні системні параметри. Наприклад, додатки використовують реєстр для зберігання відомостей, які мають бути доступними після закриття програми та при перезавантаженні додатків. Зокрема, можна зберігати налаштування кольору, положення або розмір вікна. Для різних користувачів ці відомості можуть зберігатися у різних місцях реєстру.

Клас **Registry** містить ще низку статичних властивостей, кожна з яких представляє відповідний ключ верхнього рівня:

- **Registry.ClassesRoot**
- **Registry.CurrentConfig**
- **Registry.CurrentUser**
- **Registry.DynData**
- **Registry.Users**
- **Registry.PerformanceData**

Для управління ключами в реєстрі клас **RegistryKey** визначає низку властивостей і методів. Основні з них:

- **Name:** повертає ім'я ключа реєстру
- **Close ():** закриває ключ
- **CreateSubKey ():** створює вкладений ключ, якщо він не існує
- **DeleteSubKey ():** видаляє вкладений ключ
- **DeleteValue ():** видаляє значення ключа
- **GetSubKeyNames ():** повертає колекцію імен вкладених ключів
- **GetValue ():** повертає значення ключа
- **OpenSubKey ():** відкриває вкладений ключ
- **SetValue ():** встановлює значення ключа

Створимо свій ключ у реєстрі:

Код використовує **C#** та **.NET** для створення нового ключа в реєстрі та збереження у ньому деяких значень.

Отримання доступу до ключа **HKEY_CURRENT_USER**:

```
RegistryKey currentUserKey = Registry.CurrentUser;
```

Спочатку ви отримуєте доступ до гілки реєстру **HKEY_CURRENT_USER** за допомогою **Registry.CurrentUser**.

Створення нового підключа (підключа "**HelloKey**"):

```
RegistryKey helloKey = currentUserKey.CreateSubKey("HelloKey");
```

Ви створюєте новий підключ "**HelloKey**" в гілці **HKEY_CURRENT_USER.**

Запис значень у новий підключ:

```
helloKey.SetValue("login", "admin");
helloKey.SetValue("password", "12345");
```

Ви записуєте два значення у новий підключ "**HelloKey**". Одне з іменем "**login**" і значенням "**admin**", інше з іменем "**password**" і значенням "**12345**".

Закриття підключа:

```
helloKey.Close();
```

Ви закриваєте підключ "**HelloKey**".

У результаті ви створили новий ключ в реєстрі **HKEY_CURRENT_USER** з ім'ям "**HelloKey**" та записали у нього два значення: "**login**" і "**password**".

Після цього в реєстрі ми зможемо побачити доданий ключ і два його значення:Створимо в раніше створеному ключі вкладений ключ:

RegistryKey currentUserKey = Registry.CurrentUser;

RegistryKey helloKey= currentUserKey.OpenSubKey("HelloKey", true);

RegistryKey subHelloKey = helloKey.CreateSubKey("SubHelloKey");
subHelloKey.SetValue("val", "23");
subHelloKey.Close(); helloKey.Close().

Значення **true** у виклику **OpenSubKey ("HelloKey", true);** дає змогу відкривати ключ з можливістю запису в нього з перевіркою для уникнення виникнення помилок у випадку, якщо значення відсутні або їх немає.

```
using System;
using Microsoft.Win32;

class RegistryProgram
{
    static void Main()
    {
        // Зчитування збережених в реєстрі значень
        ReadRegistryValues("HelloKey", "login", "password");

        Console.ReadLine();
    }

    static void ReadRegistryValues(string keyName, string loginValueName, string pass
    {
        try
        {
            // Отримання ключа
            RegistryKey currentUserKey = Registry.CurrentUser;
            RegistryKey helloKey = currentUserKey.OpenSubKey(keyName);
```

```
            if (helloKey != null)
            {
                // Зчитування значень
                string login = helloKey.GetValue(loginValueName)?.ToString();
                string password = helloKey.GetValue(passwordValueName)?.ToString();

                // Виведення на консоль
                Console.WriteLine($"Login: {login}");
                Console.WriteLine($"Password: {password}");

                // Закриття ключа
                helloKey.Close();
            }
            else
            {
                Console.WriteLine($"Registry key '{keyName}' not found.");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
}
```

ЗАВДАННЯ

Поділяємо студентів на групи, користуючись номерами у журналі. При цьому кожна група буде працювати над певними ключовими елементами завдання, але всі основні функції мають виконувати всі студенти.

Завдання для всіх груп:

Кожен студент, незалежно від групи, повинен реалізувати такі функції:

- *Виведення структури реєстру*: програма повинна вивести основну структуру реєстру (гілки та підгілки).
- *Створення ключа в реєстрі*: можливість створення нового ключа та редагування значень у ньому.
- *Збереження реєстрів у файл*: функція для збереження поточного стану реєстру у файл і його відновлення.
- *Застосування змін без закриття програми*: реалізувати можливість використання змін у реєстрі без перезапуску програми.

Розподіл за групами:

Група 1 (номери 1–10): фокус на **перевірці наявності ключа**.

- Додайте функціонал перевірки, чи існує ключ із вказаним шляхом. Якщо існує, виведіть відповідне повідомлення і запропонуйте вибір між створенням нового чи оновленням наявного ключа.
- **Додатково**: додайте логіку обробки помилок під час створення або оновлення ключа.

Група 2 (номери 11–20): фокус на **роботі зі значеннями в ключі**.

- Реалізуйте функцію виведення **усіх значень в ключі**, а не тільки конкретного. Програма повинна виводити список значень із зазначеного ключа реєстру.
- **Додатково**: додайте можливість **видаляти конкретні значення** в ключі, а не сам ключ.

Група 3 (номери 21–30): фокус на **графічному інтерфейсі**.

- Розширте програму, додавши **графічний інтерфейс (GUI)**, який дозволяє користувачеві працювати з реєстром через візуальні елементи. Наприклад, додайте можливість створювати, видаляти ключі, змінювати їх значення через графічні форми.
- **Додатково**: реалізуйте функцію збереження стану реєстру через інтерфейс.

Пояснення:

- **Група 1** буде зосереджена на розробці логіки перевірки і запобіганні дублювання ключів у реєстрі.
- **Група 2** працюватиме над глибшою обробкою значень ключів та їхньою модифікацією.
- **Група 3** додаватиме графічний інтерфейс, щоб зробити програму зручнішою у використанні.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке системний реєстр Windows і яка його основна функція?
2. Які ключі верхнього рівня у реєстрі Windows? Опишіть їх призначення.
3. Як можна переглянути структуру реєстру за допомогою утиліти **regedit** ?
4. Що таке клас **Registry** у просторі з іменем **Microsoft.Win32** ? Яке його призначення?

5. Як можна перевірити наявність ключа в реєстрі перед його створенням чи оновленням?
6. Який метод використання для створення нового ключа в класі **RegistryKey** ?
7. Як програмно вивести всі значення у вільному ключі реєстру?
8. Як видалити конкретне значення у ключі реєстру, не видаляючи сам ключ?
9. Які методи використовують для видалення ключів і значень у реєстрі?
10. Як можна зберегти вміст реєстру у файлі та при потребі відновити його?
11. Як виконати зміни в реєстрі без перезавантаження програми?
12. Чому важливо забезпечити перевірку ключа перед його створенням?
13. Які переваги надає графічний інтерфейс у роботі з реєстром відповідно з консольним?
14. Як можна використовувати клас **RegistryKey** для доступу до ключів реєстру?
15. Як впливають зміни в реєстрі на роботу додатків та операційної системи?

Лабораторна робота № 4

ВИКОРИСТАННЯ WINAPI ДЛЯ МАНІПУЛЮВАННЯ ВІКНАМИ ПРОГРАМ

Мета:

Навчіться використовувати функції WinAPI для маніпулювання вікнами програми на мові C++, а також ознайомтеся з методами імпорту бібліотеки та викликом функцій із зовнішніх динамічних бібліотек, таких як **user32.dll** . Засвоїти принципи роботи з функцією **FindWindow** для пошуку вікон та іншими функціями для керування їх розміром, розташуванням і станом.

ТЕОРЕТИЧНІ ВІДОМОСТІ

WinAPI (Windows Application Programming Interface) – набір функцій, наданих операційною системою Windows, що дає змогу взаємодіяти з елементами інтерфейсу, файлами, процесами та вікнами. Для роботи з вікнами програма використовує бібліотеку **user32.dll**, що надає функції для маніпуляцій вікнами: пошук, зміна розміру,розташування, закриття вікна та інше.

Типи вікон в Windows

Розрізняють кілька типів вікон у Windows, з якими можна працювати за допомогою WinAPI:

- *Основні вікна* (Main windows): це головні вікна програм, зазвичай містять панелі інструментів, меню, область для відображення вмісту.
- *Діалогові вікна* (Dialog boxes): використовуються для відображення повідомлень або отримання введення від користувача.
- *Плаваючі вікна* (Floating windows): вікна, які відображають додаткову інформацію або інструменти для основної програми.

Зазвичай основні дії з вікнами здійснюються через хендли (ручки) вікон – унікальні ідентифікатори, що їх призначає система для кожного відкритого вікна.

Архітектура повідомлень Windows

Windows використовує модель подій та повідомлень для взаємодії з програмами. Кожне вікно має свою чергу повідомлень, до якої додаються події, що відбуваються у системі або взаємодії користувача з інтерфейсом. WinAPI дозволяє відправляти повідомлення у чергу вікна через функцію SendMessage, серед яких:

- WM_CLOSE. Команда для закриття вікна.
- WM_MOVE. Вікно переміщується.
- WM_SIZE. Вікно змінює розміри.
- WM_DESTROY. Вікно знищується.
- WM_PAINT. Відповідає за перерисовку вмісту вікна.

Механізм хендлів (Handles)

Хендли (Handles) – це посилання на різні об'єкти операційної системи, включаючи вікна. У WinAPI хендли дозволяють програмам взаємодіяти з елементами ОС, такими як:

- Вікна (HWND)
- Драйвери (HDC)
- Потоки (HANDLE)

Інші корисні функції WinAPI для роботи з вікнами

- EnumWindows: перебирає всі відкриті вікна системи.
- GetWindowText: отримує текст заголовка вікна.
- GetClassName: отримує ім'я класу вікна, яке може бути корисним при пошуку вікон.
- IsWindowVisible: перевіряє, чи вікно видиме для користувача.

Права доступу та безпека

Деякі операції над вікнами, такі як закриття або переміщення, можуть вимагати підвищених прав доступу, особливо якщо вікно належить іншому процесу або користувачеві. Для цього треба враховувати привілеї додатка і забезпечити належну безпеку для уникнення конфліктів з іншими програмами.

Робота з кількома моніторами

WinAPI також дає змогу маніпулювати вікнами на системах із кількома моніторами. За допомогою функцій, таких як EnumDisplayDevices та EnumDisplayMonitors, можна отримувати інформацію про доступні монітори і змінювати розташування вікон між ними.

Практичне застосування WinAPI для автоматизації

У реальних проєктах WinAPI може використовуватися для автоматизації користувацьких дій:

- Автоматизація тестування GUI (перевірка стану вікон і взаємодія з ними).
- Інструменти керування вікнами (наприклад, автоматичне впорядкування вікон на робочому столі).
- Зміна властивостей вікон системних програм або програм інших розробників (для налаштування користувацького інтерфейсу).

Обмеження та недоліки

Хоча WinAPI є потужним інструментом для взаємодії з ОС, його використання може призвести до:

- Залежно від версії Windows, деякі функції можуть змінюватися або видалятися у новіших версіях.

- Високої складності коду при маніпуляції з великою кількістю вікон або об'єктів.
- Можливих конфліктів із системними обмеженнями, такими як обмеження прав доступу до певних вікон або процесів.

Основні функції, які будуть використовуватися у цій лабораторній роботі:

1. **FindWindow** – розмістити вікно за його класом або заголовком.
2. **SetWindowPos** – змінює положення і розмір вікна.
3. **ShowWindow** – змінює стан вікна (мінімізація, максимізація, приховування).
4. **SendMessage** – надсилає повідомлення вікну.

Опис функцій

1. **FindWindow**

✓ Призначення: Пошук вікна за класом або заголовком.

✓ **Синтаксис:**

```
HWND FindWindow(LPCSTR lpClassName, LPCSTR lpWindowName);
```

✓ **Параметри:**

- lpClassName: ім'я класу вікна (може бути NULL).
- lpWindowName: заголовок вікна (може бути NULL).

2. **SetWindowPos**

✓ Призначення: Зміна положення і розміру вікна.

✓ **Синтаксис:**

```
BOOL SetWindowPos(HWND hWnd, HWND hWndInsertAfter, int X, int Y, int cx, int cy, UINT uFlags);
```

✓ **Параметри:**

- **hWnd**: дескриптор вікна.
- **X, Y**: нові координати вікна.
- **cx, cy**: нові розміри вікна.
- **uFlags:** прапори управління позиціюванням.

3. **ShowWindow**

✓ Призначення: функція змінює стан вікна (мінімізація, максимізація, приховування).

✓ **Синтаксис:**

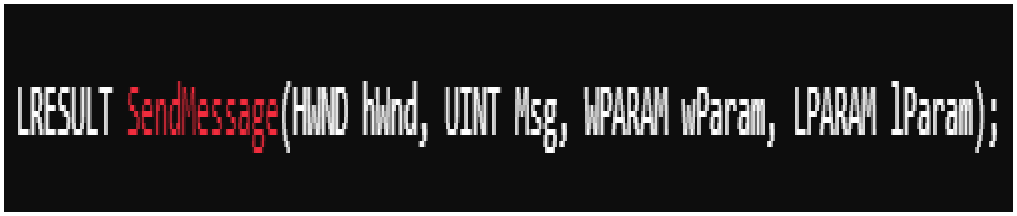
✓ **Параметри:**

- **nCmdShow** – вказує, як потрібно показати вікно (`SW_SHOW`, `SW_HIDE`, `SW_MINIMIZE`, `SW_MAXIMIZE` тощо).

4. SendMessage

✓ Призначення: дає змогу надсилати повідомлення вікну

✓ **Синтаксис:**



✓ **Параметри:**

- **Msg** – код повідомлення (наприклад, WM_CLOSE для закриття вікна).

ЗАВДАННЯ

Завдання вибираємо згідно з номером у журналі.

1. (1, 6, 11, 16) Створення власного менеджера вікон

- Ця задача навчає основ взаємодії з вікнами за допомогою WinAPI, такими як пошук вікон, зміна їх розмірів і розташування.
- Добрий початковий рівень для того, щоб навчитися використовувати такі функції, як FindWindow, SetWindowPos, MoveWindow тощо.

2. (2, 7, 12, 17) Автоматизація робочого процесу

- Автоматичне управління вікнами потребує додаткових знань, таких як робота з таймерами і розкладом, що добре підходить для поглиблення знань про синхронізацію і планування в C++.
- Використання таких функцій, як Sleep, CreateTimerQueueTimer або інших таймерів у комбінації з маніпуляцією вікнами через WinAPI.

3. (3, 8, 13, 18) Графічний редактор вікон

- Тут буде задіяно більше елементів графічного інтерфейсу, таких як створення нових вікон, маніпуляція формами, що також добре розвиває навички роботи з графікою і WinAPI.
- Це вже більш творчий підхід, що може включати виклик функцій для створення нових вікон та управління їхнім виглядом і поведінкою.

4. (4, 9, 14, 19) Спостереження за вікнами

- Це завдання потребує знань про багатозадачність або роботу з подіями, оскільки програма має автоматично відстежувати відкриті вікна та реагувати на їхні зміни.
- Тут можна використати функції, як-от EnumWindows, які допомагають знайти і перерахувати всі активні вікна, а також системні повідомлення для обробки подій.

5. **(5, 10, 15, 20) Відстеження активності користувача**
   - Це завдання орієнтоване на збереження історії роботи з вікнами, що вимагає використання додаткових механізмів запису інформації в файл або базу даних, а також відстеження часу, що буде корисним для вивчення роботи з часом і подіями в C++.
   - Можна задіяти GetForegroundWindow, GetWindowThreadProcessId для збору інформації про активні вікна.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке WinAPI і як він використовується в операційних системах Windows?
2. Яке призначення бібліотеки **user32.dll** у WinAPI?
3. Як оголошується функція із зовнішньої бібліотеки за допомогою директиви **extern "C"** у C++?
4. Для чого використовується функція **FindWindow** і як її використовувати в C++?
5. Як завантажити дескриптор вікна (HWND) у WinAPI?
6. Які параметри заміни функції **FindWindow** і що повертає у випадку успіху або невдачі?
7. Як можна змінити розмір вікна програми за допомогою WinAPI?
8. Яка функція використання для зміни розташування вікна на екрані?
9. Як можна мінімізувати, відновити або закрити вікно програми через WinAPI?
10. Які методи перевірки наявності вікна конструкції перед виконанням операцій з ним?
11. Як взаємодіяти з вікнами програми, яка не є частиною вашого процесу?
12. Що таке повідомлення Windows (повідомлення) і як працює система обробки повідомлень вікон?

13. Яка функція використовується для надсилання повідомлень між вікнами, наведіть приклад як це зробити ?
14. Як можна отримати інформацію про стан або атрибути вікна за допомогою WinAPI?
15. Що відбувається, якщо функція **FindWindow** не може знайти вікно? Як правильно обробляти таку кількість часу в програмі на C++?

Лабораторна робота № 5

ЗМІНА СИСТЕМНИХ ДАТИ ТА ЧАСУ, ТАЙМЕР ЗВОРОТНОГО ВІДЛІКУ З ВИКОРИСТАННЯ WINAPI

Мета:

Системний час – це час, який операційна система використовує для синхронізації подій, запуску процесів і роботи з додатками. У Windows системний час оновлюється системним таймером і може бути змінений вручну або програмно через API. Крім того, системний час використовується для відліку періодів часу і є критичним для таких завдань, як планування подій, таймери та будильники. Системний час у Windows визначає, коли виконуються певні завдання або коли завершуються дії, тому зміна цього часу повинна виконуватися обережно.

Основні поняття

1. **UTC і локальний час**:
    - **UTC** (Coordinated Universal Time) є загальноприйнятим стандартом для відліку часу, який не залежить від часових поясів.
    - **Локальний час** – це час для конкретної локації, який враховує часовий пояс та можливий перехід на літній або зимовий час.

Функція SetSystemTime працює з часом у форматі UTC, тому перед зміною системного часу необхідно перевести локальний час у формат UTC, якщо ви використовуєте локальний час.

2. **Привілеї для зміни системного часу**:
    - Для зміни системного часу програма повинна виконуватися від імені адміністратора, оскільки ця дія впливає на всю систему.
    - Якщо не дотриматися цього, функція SetSystemTime не буде успішною, і поверне **false**.

3. Структура SYSTEMTIME: є основною для роботи з часом у Windows API. Вона містить різні поля для збереження дати і часу, і її використання дає змогу легко працювати з різними аспектами системного часу, такими як рік, місяць, день, година, хвилини, секунди та мілісекунди.

Основні компоненти структури SYSTEMTIME:

- **wYear** – рік
- **wMonth** –місяць (від 1 до 12)
- **wDayOfWeek** – день тижня (неділя = 0, понеділок = 1 і т. д.)
- **wDay** – день місяця
- **wHour** – година (24-годинний формат)
- **wMinute** – хвилини
- **wSecond** – секунди
- **wMilliseconds** – мілісекунди

Функції Windows API для роботи з системним часом:

✓ Функція **SetSystemTime** дає змогу змінювати системний час комп'ютера. Вона працює з UTC-часом і приймає структуру **SYSTEMTIME**, яка містить нові значення дати та часу.

Синтаксис:

```
BOOL SetSystemTime(const SYSTEMTIME* lpSystemTime);
```

Параметри:

- lpSystemTime – вказівник на структуру SYSTEMTIME, яка містить нові значення для дати та часу.

lpSystemTime – вказівник на структуру SYSTEMTIME, яка містить нові значення для системного часу. Функція працює з часом у форматі UTC (Universal Time Coordinated).

- ✓ **GetSystemTime** для отримання поточного системного часу використовується функція GetSystemTime. Вона дає змогу отримати значення дати та часу в форматі UTC.

  Синтаксис:

```
void GetSystemTime(LPSYSTEMTIME lpSystemTime);
```

- ✓ **SetLocalTime** Якщо потрібно змінити локальний час, який враховує часові пояси, треба використовувати функцію **SetLocalTime**.

  Синтаксис:

```
BOOL SetLocalTime(const SYSTEMTIME* lpSystemTime);
```

Ця функція працює аналогічно SetSystemTime, але з локальним часом замість UTC.

Обмеження та рекомендації

1. **Привілеї адміністратора**: Для зміни системного часу потрібно запускати програму з правами адміністратора. В іншому випадку функція поверне помилку, і системний час не буде змінено.
2. **Коректне переведення часу**: Якщо ви працюєте з локальним часом, необхідно коректно переводити його у формат UTC перед використанням функції SetSystemTime. І навпаки, після отримання системного часу в UTC, його потрібно конвертувати у локальний для коректного відображення.
3. **Вплив на інші програми**: Зміна системного часу може впливати на інші програми, тому її треба використовувати лише при необхідності. Програми можуть некоректно обробляти події, якщо системний час змінюється під час їх роботи.

Будильник

Будильник – це програма або функція, яка виконує дію або надсилає сповіщення після настання певного часу чи після закінчення заданого інтервалу. Він може бути використаний для виконання різноманітних завдань, таких як повідомлення користувача, запуск іншої програми, відтворення звуку або виконання певної операції у системі.

Будильники часто використовують для нагадувань або автоматизації завдань, коли потрібно діяти в точно визначений час. У Windows будильники можна реалізувати через використання таймерів або функцій Windows API, таких як **SetTimer**.

Основні концепції будильника

1. **Таймери**

Таймер є важливим компонентом будильника, який дозволяє задавати інтервал часу для події. Після закінчення цього інтервалу програма викликає функцію зворотного виклику, що виконує необхідні дії.

2. **Цикл очікування**

Будильник також може працювати через цикл очікування (polling), при якому програма періодично перевіряє, чи настав певний час для виконання завдання.

3. **Відлік до події**

У деяких випадках будильник налаштовується так, щоб подія виконувалася через визначений час, наприклад, через 10 хвилин. Це робить його корисним як для коротких, так і для тривалих інтервалів.

Реалізація будильника за допомогою Windows API

Windows API пропонує кілька способів для реалізації будильника, наприклад, за допомогою функцій **SetTimer** і **Sleep**.

Функція SetTimer

Використання **SetTimer** дозволяє налаштувати таймер для виконання події після завершення інтервалу. Це асинхронний спосіб створення будильника, де функція зворотного виклику виконується після спрацьовування таймера.

Синтаксис:

```
UINT_PTR SetTimer(HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc);
```

- ✓ **hWnd**: дескриптор вікна.
- ✓ **nIDEvent**: ідентифікатор таймера.
- ✓ **uElapse**: час затримки в мілісекундах.
- ✓ **lpTimerFunc**: вказівник на функцію зворотного виклику, яка буде викликана після завершення таймера.

Функція Sleep

Для простого випадку можна використовувати функцію **Sleep**, яка зупиняє виконання програми на визначений інтервал часу.

Синтаксис:

```
void Sleep(DWORD dwMilliseconds);
```

Недоліки: Sleep зупиняє виконання програми, тому не підходить для складних завдань, де потрібно продовжувати роботу, поки будильник чекає.

Додаткові можливості будильника

1. **Звук або сповіщення**

Коли будильник спрацьовує, програма може відтворювати звукове сповіщення або відображати повідомлення. Для цього можна використати API функції для програвання звуку або відображення вікон повідомлень.

Приклад звуку через Beep

```cpp
#include <windows.h>
#include <iostream>

int main() {
    std::cout << "Будильник буде запущено через 3 секунди..." << std::endl;

    Sleep(3000);

    // Звуковий сигнал
    Beep(750, 300);

    std::cout << "Будильник спрацював!" << std::endl;

    return 0;
}
```

Таймер зворотного відліку

Таймер зворотного відліку – це механізм, який відлічує час у зворотному порядку, від заданого інтервалу до нуля. Коли таймер досягає нуля, він виконує певну дію, наприклад, відображає сповіщення, відтворює звук або викликає іншу функцію. Таймер зворотного відліку може бути використаний у різних сценаріях: від нагадувань про важливі події до управління тривалістю завдань у програмі.

Основні особливості таймера зворотного відліку

1. **Встановлення початкового часу**

Користувач або програма задає час, від якого починається відлік. Це може бути будь-який інтервал часу: від кількох секунд до кількох годин.

2. **Поступове зменшення часу**

Після запуску таймер зворотного відліку зменшує свій час у реальному часі, доки не досягне нуля.

3. **Подія після завершення**

Коли таймер досягає нуля, він викликає подію: це може бути звуковий сигнал, відображення повідомлення або виконання іншого завдання, залежно від того, як програмований таймер.

Реалізація таймера зворотного відліку за допомогою Windows API

У Windows API можна реалізувати таймер зворотного відліку за допомогою різних методів. Ось деякі з найпопулярніших способів.

Функція SetTimer дає змогу створювати таймер, який зворотно відраховує час. Таймер спрацьовує через певний інтервал, а після цього можна виконати дію або продовжити відлік.

Функція Sleep для простого таймера можна використовувати функцію Sleep, яка зупиняє виконання програми на визначений інтервал часу. У цьому випадку таймер виконується синхронно – програма чекає завершення відліку.

Приклад на C++ з використанням Sleep:

```cpp
#include <windows.h>
#include <iostream>

int main() {
    int countdown = 10;  // Початковий час для зворотного відліку

    std::cout << "Таймер зворотного відліку на 10 секунд..." << std::endl;

    // Запускаємо таймер
    while (countdown > 0) {
        std::cout << "Залишилось: " << countdown << " секунд" << std::endl;
        Sleep(1000);  // Затримка на 1 секунду
        countdown--;
    }

    std::cout << "Таймер завершено!" << std::endl;

    return 0;
}
```

ЗАВДАННЯ

Варіант 1

Завдання 1. Таймер зворотного відліку для завершення роботи

Напишіть програму, яка дозволяє встановити таймер зворотного відліку, що закриває програму або робоче вікно після завершення часу. Після запуску

таймера користувач має бачити, скільки часу залишилось, і після завершення відліку програма виводить повідомлення та закривається.

Завдання 2. Будильник з відкладенням

Створіть будильник, який дозволяє користувачеві не тільки налаштувати час спрацювання, але й функцію "відкласти" будильник на певний проміжок часу (5, 10, 15 хвилин). Після того, як будильник спрацював, користувач може вибрати варіант відкладення.

Завдання 3. Автоматична зміна системної дати і часу

Напишіть програму, яка автоматично змінює системну дату і час щодня о певній годині. Програма повинна працювати у фоновому режимі і синхронізувати системний час з інтернет-сервісом або внутрішнім планувальником.

Варіант 2

Завдання 1. Таймер зворотного відліку для перерв

Створіть програму, яка запускає таймер зворотного відліку на певний проміжок часу (наприклад, 25 хвилин для роботи). Після закінчення таймера програма сповіщає про необхідність зробити перерву на 5 хвилин і автоматично перезапускає таймер після перерви.

Завдання 2. Будильник для важливих подій

Реалізуйте будильник, який дозволяє налаштувати його спрацювання для декількох подій протягом дня (наприклад, обід, зустріч, тренування). Користувач задає кілька будильників, і програма відтворює відповідний сигнал для кожної події.

Завдання 3. Програма для ручної зміни системного часу

Створіть програму, яка дозволяє користувачеві змінювати системний час за допомогою графічного інтерфейсу. Програма повинна дозволяти вибирати дату і час у полі введення, а після підтвердження зміна повинна застосовуватись до системного часу.

Варіант 3

Завдання 1. Таймер для вправ

Напишіть програму, яка запускає таймер зворотного відліку для фітнес-вправ. Користувач задає час для виконання певних вправ (наприклад, 45 секунд на кожну вправу), після чого програма відраховує час і повідомляє про завершення.

Завдання 2. Будильник для ранкового пробудження

Створіть програму-будильник для ранкового пробудження. Користувач може вибрати час спрацювання будильника та відкладення сигналу на кілька хвилин (снуз), при цьому програма повинна відтворювати звуковий сигнал або виводити візуальне сповіщення.

Завдання 3. Таймер для зворотної зміни системного часу

Розробіть програму, яка використовує таймер для зміни системного часу на певний інтервал назад (наприклад, 5 хвилин). Програма повинна дозволяти користувачеві бачити поточний час і налаштовувати, на скільки хвилин відкласти час назад.

Варіант 4

Завдання 1. Таймер для відліку часу до завершення програми

Створіть програму, яка відраховує час до автоматичного завершення роботи програми (наприклад, через годину). Після завершення відліку програма повинна закрити всі відкриті вікна і вивести повідомлення про завершення.

Завдання 2. Будильник з вибором музичного сигналу

Розробіть програму-будильник, яка дозволяє користувачу обирати звуковий сигнал для будильника з декількох варіантів звуків. Коли будильник спрацьовує, відтворюється обраний звук, і користувач може відкласти будильник або вимкнути його.

Завдання 3. Зміна системної дати для симуляції подій

Напишіть програму, яка дозволяє користувачеві змінювати системну дату для симуляції подій у майбутньому або минулому. Наприклад, користувач

може змінити системну дату на наступний день і перевірити, як працюватимуть інші програми з цими змінами.

Варіант 5

Завдання 1. Таймер для нагадувань під час роботи

Створіть програму, яка кожні 30 хвилин відправляє сповіщення про необхідність зробити перерву або змінити положення під час роботи. Користувач задає інтервали часу, після яких буде спрацьовувати нагадування.

Завдання 2. Будильник для автоматичного завершення сесій

Напишіть програму, яка використовується для завершення роботи після певного часу. Наприклад, якщо користувач задає час 22:00, програма автоматично вимикає комп'ютер або завершує роботу після його досягнення.

Завдання 3. Зміна системної дати та часу з історією

Створіть програму, яка дозволяє не тільки змінювати системну дату і час, але й зберігає історію цих змін. Користувач може переглядати, коли і які зміни були внесені в системний час.

Варіант 6

Завдання 1. Таймер для концентрації

Створіть програму, яка запускає таймер зворотного відліку на період концентрації (наприклад, 45 хвилин). Після закінчення часу програма повинна вивести повідомлення або звуковий сигнал, нагадуючи про необхідність зробити перерву.

Завдання 2. Будильник для різних подій

Розробіть програму, яка дозволяє встановити будильники для різних подій протягом дня (наприклад, початок лекції, обід, зустрічі). Програма повинна спрацьовувати відповідно до часу подій і відтворювати різні звукові сигнали для кожної події.

Завдання 3. Зміна системної дати з автоматичним відновленням

Створіть програму, яка дозволяє користувачеві тимчасово змінювати системну дату, але автоматично відновлює реальну дату після закінчення певного періоду часу. Це може бути корисно для тестування програм.

Варіант 7

Завдання 1. Таймер для пауз у грі

Створіть програму, яка використовується під час гри для автоматичних пауз. Користувач задає інтервал часу (наприклад, кожні 20 хвилин), і після завершення таймера програма зупиняє гру або виводить повідомлення про паузу.

Завдання 2. Будильник для автоматичного запуску програм

Розробіть програму-будильник, яка не тільки відтворює звук у визначений час, але й автоматично запускає задану програму (наприклад, текстовий редактор або веббраузер) після спрацювання будильника.

Завдання 3. Таймер для тестування продуктивності системи

Створіть програму, яка змінює системний час і запускає тест продуктивності системи через певні проміжки часу. Наприклад, програма може запускати тести кожні 30 хвилин і записувати результати у файл.

Варіант 8

Завдання 1. Таймер для завершення завдання

Напишіть програму, яка встановлює таймер зворотного відліку для завершення завдання. Якщо користувач не завершив завдання до закінчення таймера, програма виводить сповіщення або зупиняє процес.

З**авдання 2. Будильник для налаштування активності**

Створіть програму-будильник, яка щогодини нагадує користувачеві про необхідність змінити вид діяльності (наприклад, зробити розминку). Користувач може задавати проміжки часу й отримувати сповіщення або звуковий сигнал.

Завдання 3. Зміна системного часу для тестування програм із обмеженням

Напишіть програму, яка змінює системний час і дозволяє тестувати інші програми, що мають часові обмеження (наприклад, тестові версії програмного забезпечення). Програма повинна відображати зміни в реальному часі.

Варіант 9

Завдання 1. Таймер для кухні

Створіть програму для кухонного таймера, яка дозволяє встановлювати кілька таймерів одночасно (для різних страв). Кожен відраховує час і виводить звуковий сигнал після завершення.

Завдання 2. Будильник для нагадування про важливі завдання

Напишіть будильник, який спрацьовує через певний час і нагадує користувачеві про важливі завдання або зустрічі. Будильник може показувати спливаюче вікно з повідомленням або відтворювати звуковий сигнал.

Завдання 3. Автоматична зміна системної дати на свята

Розробіть програму, яка автоматично змінює системну дату на певні дати (наприклад, державні свята) і запускає тематичні привітання або завдання, залежно від обраного дня.

Варіант 10

Завдання 1. Таймер для здоров'я

Створіть програму, яка нагадує користувачеві про необхідність випити воду, зробити вправи для очей або інші здорові дії через певні інтервали часу. Таймер запускається після кожного сповіщення із можливістю перенести нагадування.

Завдання 2. Будильник для автоматичного резервного копіювання

Напишіть будильник, який запускає процес автоматичного резервного копіювання файлів або баз даних у певний час. Користувач може налаштувати кілька будильників для різних днів і часу резервування.

Завдання 3. Таймер для обмеження використання програм

Розробіть програму, яка встановлює таймер на певний час, після якого закриває певні програми (наприклад, ігри або соціальні мережі), обмежуючи користувача у використанні їх після закінчення часу.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке системний час і де він використовується в операційній системі?
2. Як можна отримати поточний системний час у програмі на C++ за допомогою Windows API?
3. Яка функція використовується для зміни системного часу в Windows API? Поясніть її роботу.
4. Які особливості потрібно враховувати при зміні системної дати та часу у Windows?
5. Як впливає зміна системного часу на інші програми, що виконуються на комп'ютері?
6. Що таке будильник у контексті програмного забезпечення і як він працює?
7. Які методи можна використати для реалізації будильника в програмі на C++?
8. Опишіть процес створення таймера зворотного відліку. Як відстежувати час, що залишився?
9. Яка різниця між системним таймером та таймером зворотного відліку?
10. Як забезпечити точність таймера зворотного відліку, використовуючи Windows API?
11. Як реалізувати одночасно кілька таймерів у програмі? Наведіть приклад.
12. Які можливі сценарії використання будильника у програмуванні (наведіть три приклади)?

13. Що таке системний таймер у Windows? Які функції API можна використати для роботи з ним?
14. Як реалізувати функціонал автоматичної зміни системного часу в певний момент?
15. Які функції можна використовувати для взаємодії із системним часом у програмуванні на рівні ядра Windows?

Лабораторна робота № 6

СТВОРЕННЯ ТА ВИКОНАННЯ КОМАНДНИХ ФАЙЛІВ У WINDOWS ТА LINUX

Мета:

Навчитися створювати структуру папок і автоматизувати дії з файлуми за допомогою командних файлів (.cmd для Windows, .sh для Linux).

ТЕОРЕТИЧНІ ВІДОМОСТІ

Кроки для створення командного файлу у Windows

Notepad – це вбудований текстовий редактор операційної системи Windows, який відрізняється простотою і зручністю. Він підходить для створення та редагування текстових файлів, таких як скрипти, конфігураційні файли та інші прості документи.

Відкриття текстового редактора

1. **Вибір текстового редактора**

Для створення командних файлів використовуйте простий текстовий редактор на вашому комп'ютері. Найпопулярнішим є **Notepad** для Windows або **TextEdit** для macOS, але ви також можете використовувати будь-який інший редактор за своїм вибором (наприклад, Notepad++, Sublime Text, Visual Studio Code).

2. **Використання Notepad**

Notepad є стандартним текстовим редактором у Windows і дуже зручний для створення скриптів та конфігураційних файлів.

Щоб відкрити Notepad:

- Натисніть кнопку **"Пуск"** у нижньому лівому куті екрана.
- У полі пошуку введіть **"Notepad"**.
- Натисніть Enter, і Notepad відкриється.

3. **Робота з файлом**

У Notepad ви можете вводити текст, копіювати та вставляти фрагменти, а також зберігати файл у необхідному форматі, наприклад .cmd для командного файлу або .txt для звичайного тексту.

Збережіть файл як **"create_folders.cmd"** після того, як введете необхідні команди, описані у лабораторній роботі.

Створення нового файлу .cmd

1. **Відкриття текстового редактора**
   - Відкрийте текстовий редактор, такий як Notepad, на вашому комп'ютері.
2. **Створення нового файлу**
   - У текстовому редакторі створіть новий документ.
3. **Збереження файлу**
   - Збережіть файл з розширенням .cmd. Наприклад, ви можете назвати його create_folders.cmd.
   - Для цього виберіть **Файл** -> **Зберегти як**, у вікні, що відкриється, оберіть тип файлу **"Всі файли"** і введіть назву з розширенням .cmd.
4. **Введення команд**
   - Введіть команди, які ви хочете виконати, у текстовий файл.

Опис команд:

- ✓ @echo off: вимикає відображення команд у вікні.
- ✓ REM: коментар для пояснення дій.
- ✓ if not exist: перевіряє, чи існує папка, і створює її, якщо вона відсутня.
- ✓ mkdir: створює нову папку.
- ✓ copy: копіює файли з однієї папки до іншої.
- ✓ for /L: цикл для виведення чисел з кроком.
- ✓ start: відкриває файли в асоційованій програмі.
- ✓ pause: призупиняє виконання для перегляду результатів.

Виконання командного файлу

Після створення і збереження файлу .cmd, ви можете запустити його, двічі клацнувши на ньому в Провіднику Windows або ввівши його ім'я у командному рядку (Command Prompt або PowerShell) та натиснувши Enter.

Можна реалізувати функціональність створення дерева папок за допомогою C++ для операційних систем Windows і Linux, використовуючи системні виклики або бібліотеки.

Наприклад:

Опис коду

- **#include <filesystem>**: підключає бібліотеку для роботи з файловою системою (доступна в стандарті C++17).
- **namespace fs = std::filesystem;**: створює псевдонім для зручності.
- **fs::exists(mainFolder)**: перевіряє, чи існує папка.
- **fs::create_directory(mainFolder)**: створює папку.
- **fs::create_directory(mainFolder + "/SubFolder1")**: створює підпапки.

```cpp
#include <iostream>
#include <filesystem>  // Доступно в C++17 і вище

namespace fs = std::filesystem;

int main() {
    // Назва основної папки
    std::string mainFolder = "MainFolder";

    // Перевірка наявності папки "MainFolder"
    if (!fs::exists(mainFolder)) {
        // Створення основної папки
        fs::create_directory(mainFolder);
        std::cout << "Папка '" << mainFolder << "' створена." << std::endl;

        // Створення підпапок
        fs::create_directory(mainFolder + "/SubFolder1");
        fs::create_directory(mainFolder + "/SubFolder2");
        fs::create_directory(mainFolder + "/SubFolder3");

        std::cout << "Підпапки успішно створені." << std::endl;
    } else {
        std::cout << "Папка '" << mainFolder << "' вже існує." << std::endl;
    }

    return 0;
```

Кроки для створення командного файлу у Linux.

Visual Studio Code – це безкоштовний редактор коду, розроблений Microsoft. Він надає підтримку для численних мов програмування, таких як Python, JavaScript, C++, C#, Java, і багато інших. Однією з ключових переваг VS Code є його розширювальність: користувачі можуть додавати нові функції та підтримувати мови програмування через розширення.

Основні характеристики

1. *Інтеграція з системами контролю версій*: включає підтримку Git, що дозволяє зручно управляти версіями коду.
2. *Набір потужних функцій*: такі як автозавершення коду, підсвічування синтаксису, можливість інтеграції терміналу, дебагер та інші.
3. *Розширення та теми*: можливість встановлення різних розширень та налаштування інтерфейсу.
4. *Кросплатформеність**:*** доступний на Windows, macOS і Linux.

Як встановити Visual Studio Code в Linux

Встановлення через пакетний менеджер

- ✓ Для Ubuntu або Debian виконайте такі команди в терміналі:

```
sudo apt update
sudo apt install code
```

- ✓ Для Fedora або Red Hat:

```
sudo dnf install code
```

Завантаження з офіційного сайту

- ✓ Перейдіть на офіційний сайт VS Code.
- ✓ Виберіть версію для Linux, завантажте та встановіть її.

Як створити та редагувати командний файл

1. **Створення нового файлу**:
   - Відкрийте VS Code.
   - У верхньому меню виберіть "File" -> "New File" або натисніть Ctrl + N.
2. **Написання скрипта**:
   - Введіть команди, які ви хочете виконати, наприклад:

```
#!/bin/bash
echo "Створюємо папки"
mkdir -p ~/main/SubFolder1 ~/main/SubFolder2
echo "Папки створено!"
```

3. **Збереження файлу**:
   - Виберіть "File" -> "Save As..." або натисніть Ctrl + S.
   - Введіть назву файлу, наприклад, create_folders.sh, і виберіть тип файлу "All Files" для збереження з розширенням .sh.
4. **Зробіть файл виконуваним**:

   Відкрийте термінал у VS Code (використовуючи Ctrl +) і введіть:

```
chmod +x ~/path/to/create_folders.sh
```

5. **Запуск скрипта**:

   У терміналі введіть:

```
~/path/to/create_folders.sh
```

   Ви побачите вивід у терміналі, який підтверджує виконання команд.

Опис команд:

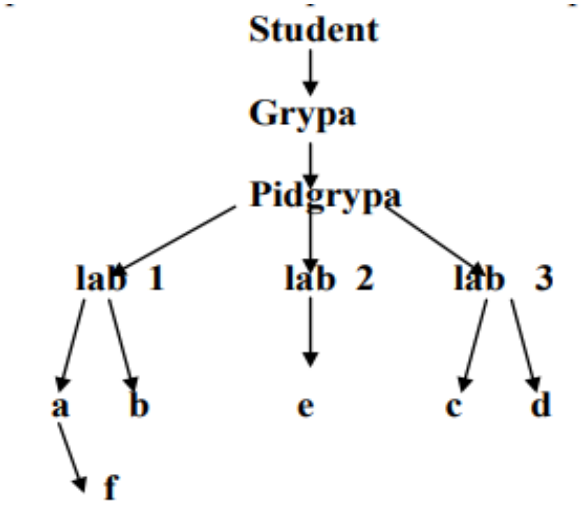
- ✓ **mkdir -p**: створює папку (включаючи всі батьківські папки, якщо вони не існують).

- ✓ **cp:** копіює файли з однієї папки до іншої.
- ✓ **for**: цикл для виведення чисел з кроком.
- ✓ **find**: шукає файли з певним розширенням і відкриває їх за допомогою xdg-open.

ЗАВДАННЯ

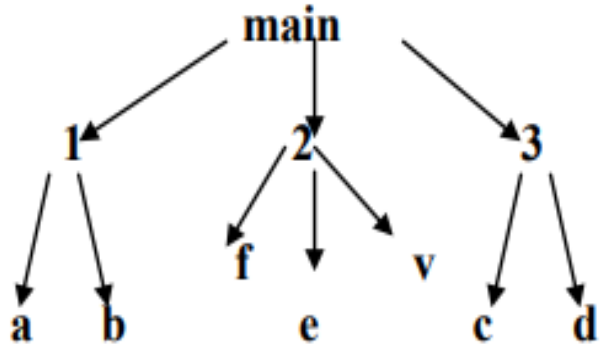
Варіант 1

1. Створення нового текстового документа
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 7 до 137 з кроком 8
4. Відкриття всіх файлів з розширенням **.doc** на диску **C**
5. Копіювання файлів із розширенням **.txt** з папки **main1** в папку **bin**
6. Збереження файлу **(**для Windows і для Linux)
   - ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
   - ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 2

1. Створення нового текстового документа з розширенням. **txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 1 до 100 з кроком 5
4. Відкриття всіх файлів з розширенням **.txt** на диску **C**
5. Копіювання файлів із розширенням .txt з папки main1 в папку bin
6. Збереження файлу **(**для Windows і для Linux)
   - ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
   - ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).

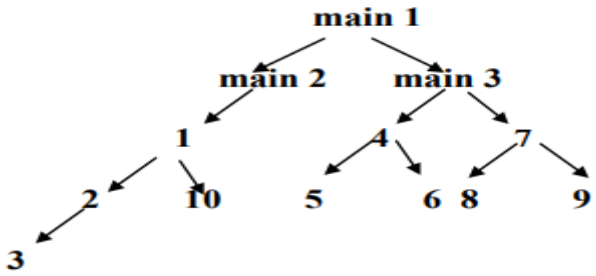


Варіант 3

1. Створення нового текстового документа з розширенням. txt.
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 20 до 85 з кроком 5
4. Відкриття всіх файлів із розширенням . exe на диску C
5. Копіювання файлів із розширенням . exe з папки main1 в папку bin

6. Збереження файлу (для Windows і для Linux)

- ✓ Збережіть файл із розширенням .cmd (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 4

1. Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

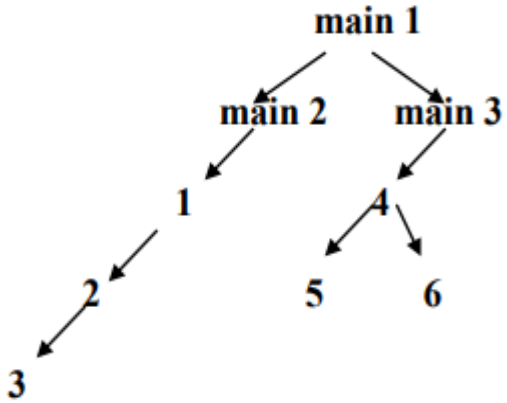
3. Цикл для виведення діапазону чисел від 20 до 1000 з кроком 15

4. Відкриття всіх файлів із розширенням **. txt** на диску **C**

5. Копіювання файлів із розширенням . txt з папки main1 в папку bin

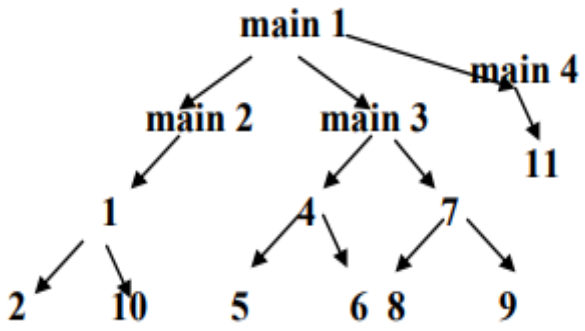
6. Збереження файлу **(**для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 5

1. Створення нового текстового документа з розширенням. **Txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 202580 до 3256987 з кроком 12
4. Відкриття всіх файлів із розширенням .exe на диску C
5. Копіювання файлів із розширенням . exe з папки main1 в папку bin
6. Збереження файлу **(**для Windows і для Linux)
   - ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
   - ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).

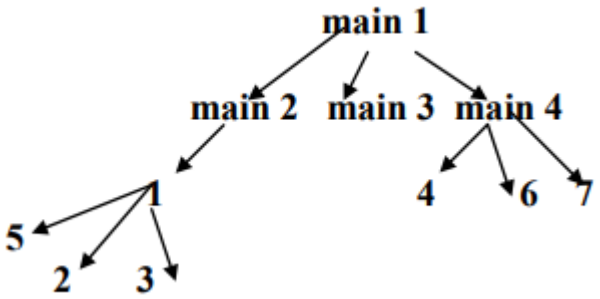


Варіант 6

1. Створення нового текстового документа з розширенням. **Txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 120 до 12000 з кроком 105
4. Відкриття всіх файлів із розширенням . txt на диску C
5. Копіювання файлів із розширенням . txt з папки main1 в папку bin

6. Збереження файлу (для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 7

1. Створення нового текстового документа з розширенням. Txt.

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

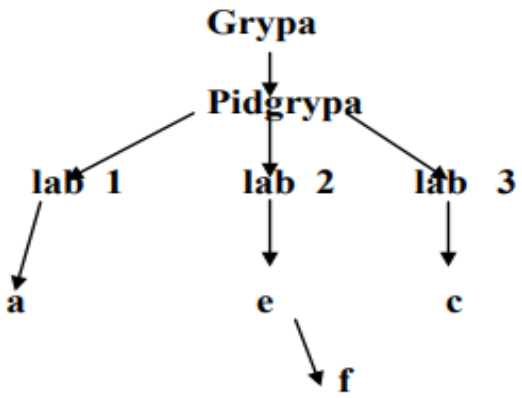
3. Цикл для виведення діапазону чисел від 10000 до 309887 з кроком 126

4. Відкриття всіх файлів із розширенням .ipg на диску С

5. Копіювання файлів із розширенням . ipg з папки main1 в папку bin

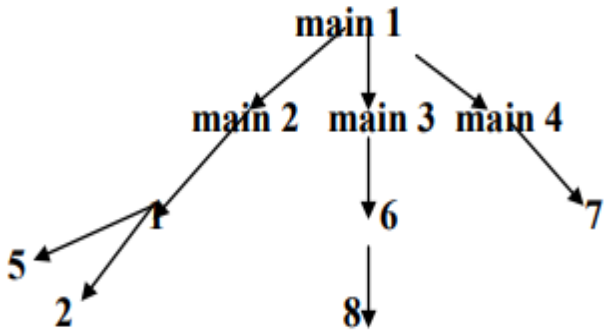
6. Збереження файлу (для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 8

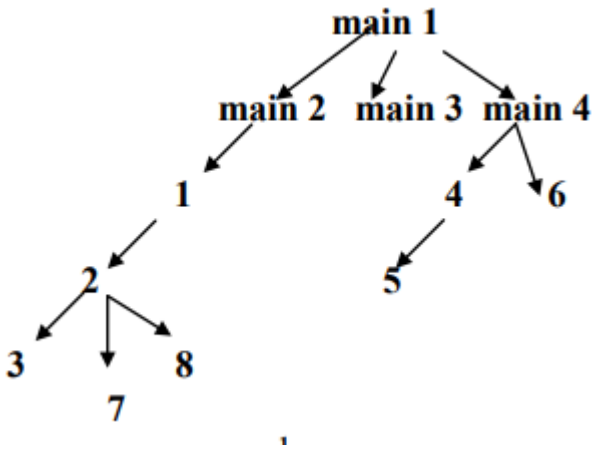
1. Створення нового текстового документа з розширенням. **txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 12580 до 32987 з кроком 23
4. Відкриття всіх файлів із розширенням .exe на диску C
5. Копіювання файлів із розширенням . exe з папки main1 в папку bin
6. Збереження файлу **(**для Windows і для Linux)
   - ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
   - ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 9

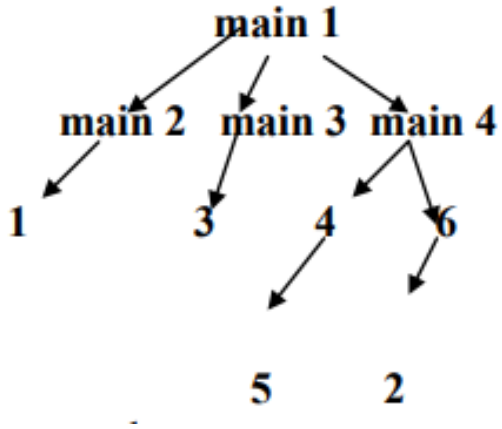
1. Створення нового текстового документа з розширенням. **txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ і запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 10000 до 309887 з кроком 126
4. Відкриття всіх файлів із розширенням .ipg на диску C
5. Копіювання файлів із розширенням . ipg з папки main1 в папку bin
6. Збереження файлу (для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 10

1. Створення нового текстового документа з розширенням. **txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 1350 до 9887 з кроком 46
4. Відкриття всіх файлів із розширенням .exe на диску C
5. Копіювання файлів із розширенням . exe з папки main1 в папку bin
6. Збереження файлу (для Windows і для Linux)
   - ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
   - ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 11

1. Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

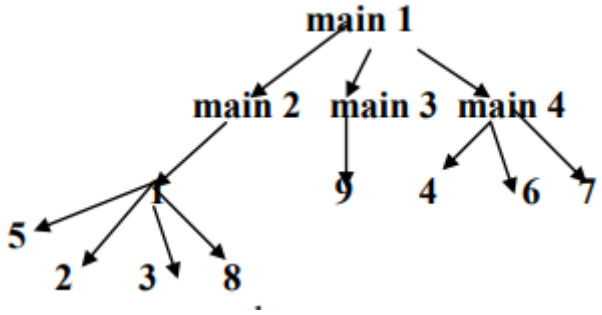
3. Цикл для виведення діапазону чисел від 10000 до 309887 з кроком 126

4. Відкриття всіх файлів із розширенням .ipg на диску C

5. Копіювання файлів із розширенням . ipg з папки main1 в папку bin

6. Збереження файлу (для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 12

Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

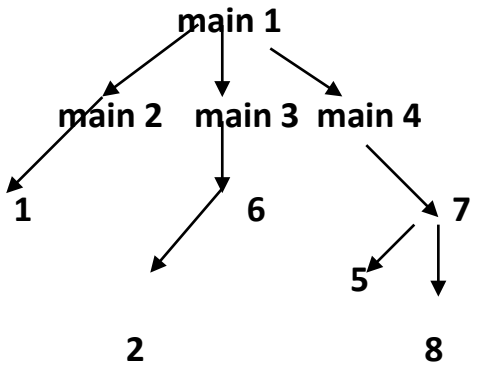
3. Цикл для виведення діапазону чисел від 22590 до 83595 з кроком 35

4. Відкриття всіх файлів із розширенням **. exe** на диску **C**

5. Копіювання файлів із розширенням **. exe** з папки **main1** в папку **bin**

6. Збереження файлу **(**для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 13

1. Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

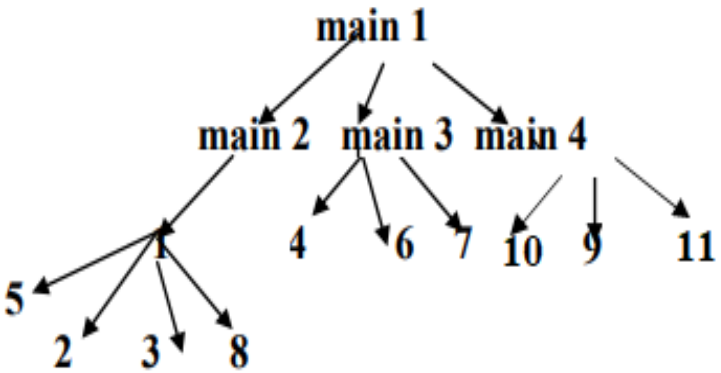
3. Цикл для виведення діапазону чисел від 2200 до 3087 з кроком 26

4. Відкриття всіх файлів із розширенням .ipg на диску C

5. Копіювання файлів із розширенням . ipg з папки main1 в папку bin

6. Збереження файлу (для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 14

1. Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

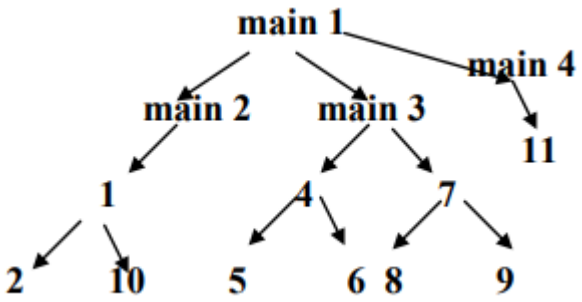
3. Цикл для виведення діапазону чисел від 20 до 1000 з кроком 15

4. Відкриття всіх файлів із розширенням . txt на диску C

5. Копіювання файлів із розширенням . txt з папки main1 в папку bin

6. Збереження файлу **(**для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 15

1. Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

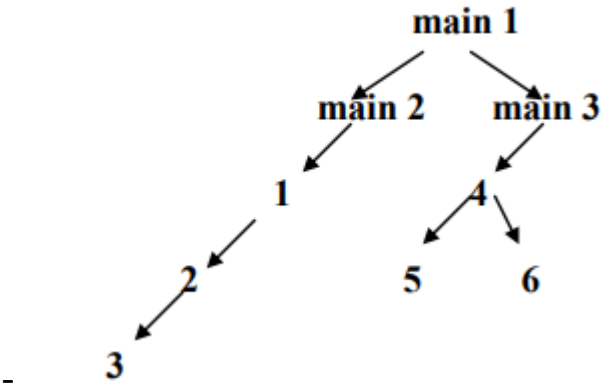
3. Цикл для виведення діапазону чисел від 202580 до 3256987 з кроком 12

4. Відкриття всіх файлів із розширенням .exe на диску C

5. Копіювання файлів із розширенням . exe з папки main1 в папку bin

6. Збереження файлу (для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



-

.

Варіант 16

1. Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

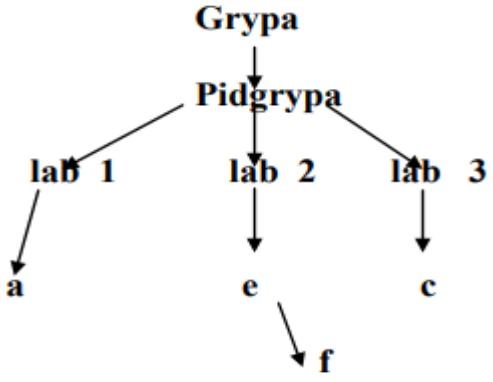
3. Цикл для виведення діапазону чисел від 120 до 12000 з кроком 105

4. Відкриття всіх файлів із розширенням **. txt** на диску **C**

5. Копіювання файлів із розширенням **. txt** з папки **main1** в папку **bin**

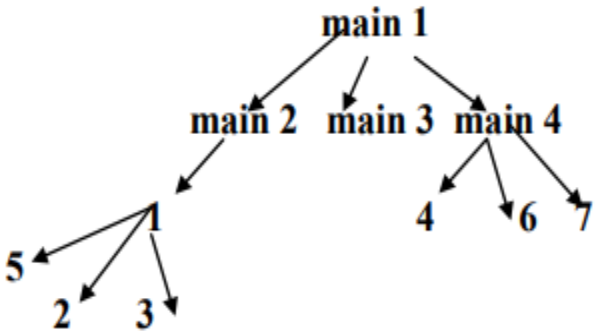
6. Збереження файлу **(**для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 17

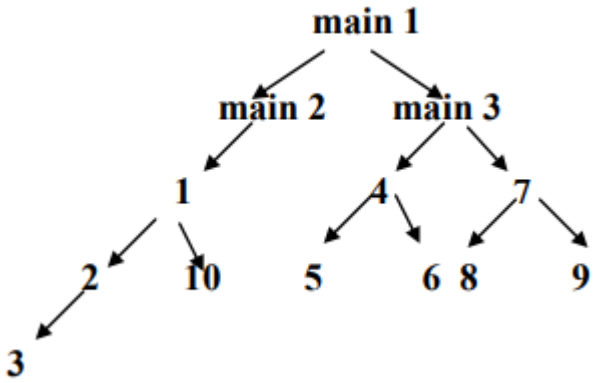
1. Створення нового текстового документа з розширенням **txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 10000 до 309887 з кроком 126
4. Відкриття всіх файлів із розширенням .ipg на диску C
5. Копіювання файлів із розширенням . ipg з папки main1 в папку bin
6. Збереження файлу (для Windows і для Linux)
   - ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
   - ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 18

1. Створення нового текстового документа з розширенням. **txt.**
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 1 до 100 з кроком 5
4. Відкриття всіх файлів із розширенням .txt на диску C
5. Копіювання файлів із розширенням .txt з папки main1 в папку bin
6. Збереження файлу **(**для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 19

1. Створення нового текстового документа з розширенням. **txt.**

- ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
- ✓ Створіть новий документ та запишіть необхідний набір команд.

2. Запис команд для створення дерева папок

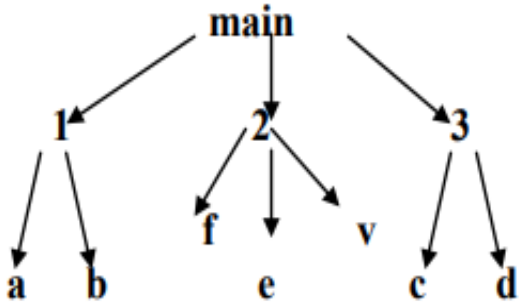
3. Цикл для виведення діапазону чисел від 20 до 85 з кроком 5

4. Відкриття всіх файлів із розширенням **. exe** на диску **C**

5. Копіювання файлів із розширенням **. exe** з папки **main1** в папку **bin**

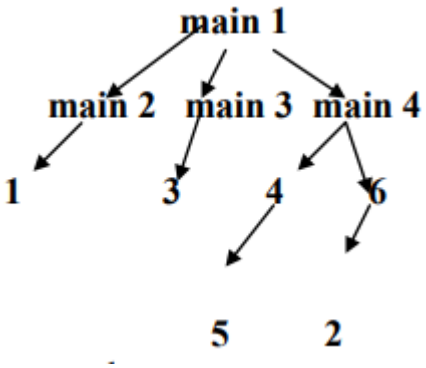
6. Збереження файлу **(**для Windows і для Linux)

- ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
- ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



Варіант 20

1. Створення нового текстового документа з розширенням. Txt.
   - ✓ Відкрийте текстовий редактор (наприклад, Notepad для Windows або nano/gedit для Linux).
   - ✓ Створіть новий документ та запишіть необхідний набір команд.
2. Запис команд для створення дерева папок
3. Цикл для виведення діапазону чисел від 1350 до 9887 з кроком 46
4. Відкриття всіх файлів із розширенням .exe на диску С
5. Копіювання файлів із розширенням . exe з папки main1 в папку bin
6. Збереження файлу **(**для Windows і для Linux)
   - ✓ Збережіть файл із розширенням **.cmd** (наприклад, create_structure.cmd).
   - ✓ Збережіть файл із розширенням **.sh** (наприклад, create_structure.sh).



КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яке призначення командного файлу (.cmd) в операційній системі Windows?
2. Які текстові редактори можна використовувати для створення командних файлів на Windows та Linux?
3. Які команди необхідно включити до командного файлу для створення структури папок?

4. Як зберегти файл із розширенням .cmd у текстовому редакторі?
5. Що означає команда @echo off у командному файлі?
6. Як за допомогою циклу for вивести діапазон чисел від 7 до 137 з кроком 8?
7. Яка команда використовується для відкриття всіх файлів із розширенням .doc на диску C?
8. Якою командою можна скопіювати файли з розширенням .txt з папки main1 в папку bin?
9. Які переваги має використання && для послідовних команд у командному файлі?
10. Як можна перевірити, чи існує папка перед її створенням?
11. Які основні команди для створення папок можна використовувати в Linux?
12. Як кожна реалізувати динамічне створення папок у C++?
13. Як можна додати логування дій програми у вашому скрипті?
14. Які інструменти або бібліотеки можна використовувати для створення графічного інтерфейсу для програми?
15. Як ви можете адаптувати ваш скрипт для роботи в різних операційних системах, таких як Windows та Linux?

Лабораторна робота № 7

РОБОТА З ЖОРСТКИМИ ДИСКАМИ

Мета:

Ознайомити студентів з основними операціями, пов'язаними з жорсткими дисками в операційній системі Windows, а також надати їм практичні навички управління дисками та файловою системою.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Жорсткий диск (HDD) є невід'ємною частиною системного блоку комп'ютера та відіграє ключову роль у зберіганні всієї інформації користувача.

Основні функції жорсткого диска:

1. **Зберігання операційної системи та програмного забезпечення**. Жорсткий диск містить операційну систему та різноманітне програмне забезпечення, таке як драйвери для пристроїв, текстові редактори, веббраузери тощо.
2. **Зберігання особистих файлів користувача**. Користувачі зберігають на жорсткому диску особисті файли, які зазвичай займають багато місця та розподілені по різних розділах і кластерах.
3. **Організація інформації на диску**. Інформація на жорсткому диску організована у вигляді кластерів. Кластер – це найменша одиниця зберігання на диску, і весь файл може займати один або більше кластерів залежно від розміру.
4. **Віддалене зберігання та резервне копіювання**. З метою забезпечення безпеки даних, користувачі також можуть зберігати свої файли на зовнішніх носіях або створювати резервні копії на інших пристроях.

Ці функції дають змогу користувачам ефективно управляти своїми даними та забезпечувати безпеку і доступність інформації на жорсткому диску.

У Windows 10 використовується нова модель управління дисками, яка базується на розділах **GPT** (GUID Partition Table). У цій операційній системі жорсткий диск може бути ініціалізований як базовий або динамічний.

Базовий диск (Basic Disk)

Базовий диск – це традиційний тип диска, який підтримує стандартні розділи і файлові системи, такі як FAT, FAT32, а також NTFS. У Windows 10 базовий диск містить основні розділи, додаткові розділи та логічні диски.

Для створення базових дисків і роботи з ними можна скористатися програмою "Диспетчер дисків" або використовувати командний рядок (за допомогою команди diskpart). Основні розділи на базових дисках можуть використовуватися для завантажувальних операційних систем, основних даних тощо.

На базовому диску можуть бути створені прості томи (simple volumes), які складаються з одного фізичного диска або декількох без використання складних механізмів, таких як дзеркальність або резервне копіювання.

Динамічний диск (Dynamic Disk)

Динамічний диск – це розширений тип диска, який підтримує складені томи (spanned volumes), дзеркальні томи (mirror volumes), томи з резервними копіями (RAID-5 volumes) та інші розширені функції. Динамічні диски дають змогу виконувати додаткові функції, такі як динамічні резервні копії, динамічне розширення об'єднаних томів тощо.

Базові диски можна конвертувати у динамічні та налаштовувати їх за допомогою інструментів управління дисками Windows 10. Загалом, поняття базових та динамічних дисків залишаються актуальними в Windows 10, і вони використовуються для організації та управління дисками і томами даних на вашому комп'ютері.

Важливі зауваження

Типи запам'ятовувальних пристроїв не пов'язані з типом файлової системи. Дискова система може містити такі пристрої різного типу в будь-якій комбінації. Однак кожен диск або том на диску повинен використовувати один і той самий тип збережувального пристрою.

Типи томів у системах управління дисками

Існує кілька типів томів, які використовуються у системах управління дисками, таких як Windows. Ось деякі з них і їхні характеристики.

1. Простий том (Simple Volume)

- Це базовий тип тому, який використовує один фізичний диск або його частину.
- Простий том може містити лише один том і не підтримує резервне копіювання даних та інші функції, притаманні динамічним томам.

2. Динамічний простий том (Dynamic Simple Volume)

- Це аналог простого том у у динамічному об'ємі.
- Як і простий том, він використовує один фізичний диск або частину фізичного диска, але може бути частиною динамічного об'єму.

3. Динамічний об'єм (Dynamic Volume):

- Динамічний об'єм дає змогу створювати складніші конфігурації томів, такі як складені томи, дзеркальні томи, томи з резервними копіями тощо.
- Вони можуть містити кілька фізичних дисків або частин фізичних дисків і надають різні функції захисту даних.

4. Дзеркальний том (Mirror Volume)

- Дзеркальний том створює копію даних на іншому фізичному диску з метою забезпечення надійності і резервного копіювання даних.
- Він дозволяє продовжувати роботу з даними навіть у випадку відмови одного з дисків.

5. Том з резервною копією (Spanned Volume)

- Розділ з резервною копією об'єднує два або більше фізичних дисків у єдиний том без дублювання даних.
- Це дозволяє збільшити обсяг доступного простору для зберігання даних, але не забезпечує надійності, як дзеркальний том.

Віртуальна машина (ВМ) – це програмна емуляція комп'ютерної системи, яка дає змогу запускати окремі операційні системи (ОС) на фізичному комп'ютері. Віртуальні машини здійснюють віртуалізацію для створення середовища, де програми можуть працювати так, ніби вони не запущені на окремому комп'ютері. Ось декілька ключових аспектів.

Основні характеристики віртуальних машин

1. **Віртуалізація**. Віртуальна машина використовує гіпервізор – програму, яка створює і керує ВМ. Гіпервізор може працювати як на базовій ОС (тип 2), так і на апаратному забезпеченні (тип 1).

2. **Ізоляція**. Віртуальні машини ізольовані одна від однієї, а це означає, що при умові розпізнання однією ВМ збоїв або атак, інші не будуть на це впливати.

3. **Апаратні ресурси**. ВМ використовує частину ресурсів фізичного комп’ютера, таких як процесор, оперативна пам’ять та диск, але ці ресурси можуть бути динамічно налаштовані залежно від потреби.

4. **Гнучкість.** Віртуальні машини не можна легко встановлювати, видаляти та переміщувати ОС і додатки, які спрощують тестування нових програм або налаштувань.

5. **Можливість знімків**. Багато гіпервізорів можуть створити знімки стану ВМ, що дає можливість повернутися до попереднього стану у разі виникнення проблеми.

Перед початком роботи з віртуальною машиною з операційною системою Windows переконайтеся, що ваш комп’ютер відповідає таким мінімальним системним вимогам:

1. **Процесор**: 1 ГГц або швидкий з підтримкою віртуалізації (Intel VT-x або AMD-V).

2. **Оперативна пам'ять (RAM)**: мінімум 4 ГБ (рекомендується 8 ГБ або більше для кращої продуктивності).

3. **Вільне місце на жорсткому диску**: принаймні 20 ГБ вільного місця для встановлення Windows і віртуальної машини. Рекомендується більше місць для створення та зберігання розділів.

4. **Відеокарта**: підтримка DirectX 9 або новішої версії з драйверами WDDM.

5. **Віртуалізаційне програмне забезпечення**: встановлене програмне забезпечення для віртуалізації (наприклад, VMware, VirtualBox або Hyper-V).

Для розділення накопичувача на локальні диски у Windows 10 за допомогою сторонніх програм для управління дисками, які надають широкі можливості для розділення, зміни розміру, об'єднання та інші операції з дисками і розділами.

Деякі з них включають:

- **EaseUS Partition Master**. Це потужне програмне забезпечення з інтуїтивно зрозумілим інтерфейсом, яке дозволяє виконувати різноманітні операції з розділами, такі як зміна розміру, переміщення, копіювання та багато іншого.
- **MiniTool Partition Wizard**. Ще один добре відомий інструмент для управління розділами, який має широкий набір функцій, включаючи конвертацію файлових систем, відновлення втрачених розділів, перенесення операційних систем і багато іншого.
- **Acronis Disk Director**. Цей продукт відомий надійністю і функціональністю. Він дає змогу проводити різноманітні операції з розділами, а також має додаткові інструменти для управління дисками, такі як клонування та відновлення.
- **AOMEI Partition Assistant**: Ця програма має інтуїтивний і простий у використанні інтерфейс та забезпечує широкі можливості для управління

розділами, включаючи розділення, зміну розміру, конвертацію файлових систем та інші операції.

Ці програми можуть бути корисними, особливо якщо вам потрібно виконати складні операції з дисками або якщо вам просто зручніше користуватися графічним інтерфейсом, ніж командним рядком. Перед використанням будь-якої програми завжди рекомендується резервне копіювання важливих даних, особливо при роботі з розділами і файловими системами.

ЗАВДАННЯ

Виконайте такі дії:

1. За допомогою меню керування дисками:

- Створіть новий основний та додатковий розділ на жорсткому диску та відформатуйте його під файл у системі NTFS.
- Змініть літеру основного розділу (зауважте, що змінювати літеру системного розділу не коректно).
- Відкрийте створений розділ і звільніть простір на диску.
- Змініть розмір розділу, збільшивши його на певну кількість мегабайтів.

2. За допомогою командного рядка:

- Створіть кілька логічних розділів. Один із них підключіть як NTFS-папку.
- Змініть розмір логічного розділу.
- Видаліть основний розділ.
- Перетворіть диск на динамічний (зверніть увагу, що цей процес не повертається без втрати даних).

3. За допомогою сторонніх програм для управління дисками:

- Створіть простий том, залишаючи нерозподіленою деяку область.
- Розширте простий том, відформатуйте його та видаліть.
- Створіть складений том, залишаючи нерозподіленою деяку область.
- Розширте складений том, відформатуйте його та видаліть. Створіть ще один складений том, підключивши його як NTFS-папку.
- Створіть том із відвідуванням, залишаючи нерозподіленою деяку область.
- Розширте том із відвідуванням, відформатуйте його та видаліть.
- Створіть дзеркальний том, залишаючи нерозподіленою деяку область.
- Розширте дзеркальний том, відформатуйте його та видаліть.

Рекомендації:

- Резервне копіювання: завжди робіть резервну копію важливих даних перед виконанням змін до розділів та томів.
- Ознайомтеся з термінами: якщо ви не знайомі з термінами, такими як "складений том" або "том з відвідуванням", обов'язково ознайомтеся з їх значенням перед виконанням завдання.
-

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке базовий диск у Windows? Які його основні характеристики?
2. Які типи розділів підтримує базовий диск?
3. Яка команда в командному рядку використовується для управління дисками та розділами?
4. Які основні переваги використання динамічного диска відповідають базовому диску?

5. Які функції надають дзеркальні томи? Чому їх рекомендують використовувати?

6. Яка різниця між простим томом і складеним томом?

7. Що таке резервування та які типи резервування можна використовувати на дисках?

8. Як можна змінити літеру розділу за допомогою меню управління дисками?

9. Які дії потрібно виконати для видалення розділу та звільнення місця на диску?

10. Яка команда в командному рядку використовується для створення нового розділу на диску?

11. Що таке логічний розділ і як його можна підключити як NTFS-папку?

12. Що таке динамічний диск і які його переваги в порівнянні з базовим?

13. Які програми для управління дисками можна використовувати, крім стандартного меню Windows?

14. Як можна розширити простий том та які вимоги до цього?

15. Які дії потрібно виконати для перетворення базового диска на динамічний без втрати даних?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Основна:

1. Зайцев В.Г. Операційні системи [Електронний ресурс] : навчальний посібник для студентів спеціальності 123 «Комп'ютерна інженерія» / В.Г. Зайцев, І.П. Дробязко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 2,22 Мбайт). Київ : КПІ ім. Ігоря Сікорського, 2019. 240 с. – Назва з екрана.
2. Микитишин А.Г. Операційні системи : консп. лекц. / укл. А.Г. Микитишин, І.В. Чихіра. Тернопіль : ТНТУ імені Івана Пулюя, 2016. 107 с.
3. Погребняк Б.І. Операційні системи : навч. посібник / Б.І.Погребняк, М.В.Булаєнко: Харків. нац. ун-т міськ. госп-ва ім. О.М. Бекетова. Харків : ХНУМГ ім. О.М. Бекетова, 2018. 104 с.
4. Федотова-Півень І.М. Операційні системи : навчальний посібник /за ред. В.М. Рудницького ; І.М. Федотова-Півень, І.В. Миронець, О.Б. Півень, С.В. Сисоєнко, Т.В. Миронюк ; Черкаський державний технологічний університет. Харків : ТОВ «ДІСА ПЛЮС», 2019. 216 с.
5. Шеховцов В.А. Операційні системи : підручник для вузів. Київ : Видавнича група BHV, 2015. 575 с.

Додаткова література:

1. Спірідонов В.І., Войтков В.Г. Обчислювальна техніка і програмування. Хмельницький : ХТІ, 2008. 374 с.
2. Бондаренко М.Ф., Качко О.Г. Операційні системи : навчальний посібник. Харків : Компанія СМІТ, 2008. 432 с.

Електронне навчально-методичне видання

Ольга ГАРБИЧ-МОШОРА, Христина ВОЙТОВИЧ

Операційні системи

122 «Комп'ютерні науки»

Дрогобицький державний педагогічний університет
імені Івана Франка

Редактор
Ірина Невмержицька
Технічний редактор
ІринаАртимко

Здано до набору 03.12.2024 р. Формат 60х90/16. Гарнітура Times. Ум. друк. арк. 5,25. Зам. 111.

Дрогобицький державний педагогічний університет імені Івана Франка. (Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виготівників та розповсюджувачів видавничої продукції ДК № 5140 від 01.07.2016 р.). 82100, Дрогобич, вул. Івана Франка, 24, к. 203