

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра прикладної математики

ПРОГРАМУВАННЯ ТА ВИКОРИСТАННЯ DLL В ОС WINDOWS

методичні рекомендації до виконання лабораторної роботи №5

із дисципліни:

«Операційні системи»

Київ — 2015

## Зміст

Вступ .....	3
1 Постановка задачі .....	4
2 Теоретичні відомості.....	6
2.1 Загальна інформація про бібліотеки динамічного зв'язування ..	6
2.1.1 Динамічне зв'язування та його відмінності від стати- чного зв'язування .....	6
2.1.2 Типи функцій бібліотек динамічного зв'язування .....	8
2.1.3 Два типи динамічного зв'язування .....	9
2.2 Створення бібліотеки динамічного зв'язування .....	11
2.2.1 Складові бібліотеки динамічного зв'язування .....	11
2.2.2 Визначення експортованих функцій за допомогою мо- дифікаторів .....	12
2.2.3 Визначення експортованих функцій за допомогою фай- лу визначення модуля .....	13
2.2.4 Функція точки входу .....	14
2.3 Використання бібліотек динамічного зв'язування в аплікаціях .	16
2.3.1 Використання бібліотек неявного динамічного зв'язування	16
2.3.2 Використання бібліотек явного динамічного зв'язування .	19
3 Порядок здачі лабораторної роботи та вимоги до звіту .....	23
Перелік посилань .....	24

## ВСТУП

Особливістю операційної системи (ОС) Windows є використання *бібліотек динамічного зв'язування* (dynamic link libraries, DLL). Бібліотека динамічного зв'язування — це модуль, що містить функції та дані, які можуть використовувати інші модулі (аплікації чи інші бібліотеки). Іншими словами, DLL — це виконуваний файл, який відіграє роль розділюваної бібліотеки функцій та даних.

За допомогою динамічного зв'язування процеси можуть викликати функції, які не є складовою частиною їхнього коду. Це дозволяє різним аплікаціям використовувати спільний код, а також сприяє модулярності коду та його розширюваності. У свою чергу, це спрощує модифікацію функціональності аплікацій та полегшує повторне використання коду.

ОС Windows сама по собі складається переважно з DLL, і будь-яка аплікація, яку написано з використанням Windows API, використовує можливості DLL.

У даній роботі розглядатимемо засоби роботи з бібліотеками динамічного зв'язування в ОС Windows версії 2000 та вище.

## 1 ПОСТАНОВКА ЗАДАЧІ

У даній лабораторній роботі потрібно ознайомитися з концепцією динамічного зв'язування в ОС Windows, навчитися створювати за допомогою засобів Windows API бібліотеки динамічного зв'язування та використовувати їх в аплікаціях.

У рамках виконання лабораторної роботи потрібно:

а) ознайомитися з теоретичними відомостями, викладеними в розділі 2;

б) написати будь-якою мовою програмування бібліотеку динамічного зв'язування, що реалізує певні функції; під час написання бібліотеки не можна використовувати засоби MFC;

в) написати будь-якою мовою програмування програму, яка повинна демонструвати використання написаної бібліотеки динамічного зв'язування:

1) за допомогою явного зв'язування;

2) за допомогою неявного зв'язування;

програма повинна використовувати тільки засоби Windows API і не повинна використовувати засоби MFC;

г) відлагодити програму в ОС Windows версії 2000 та вище;

д) підготувати звіт із лабораторної роботи відповідно до вимог розділу 3.

Кожен студент повинен написати бібліотеку, що реалізує індивідуальний набір функцій. Бібліотеки різних студентів не можуть реалізовувати однакові функції. Кількість функцій в одній бібліотеці не може бути

меншою за 5.

Перелік типових бібліотек динамічного зв'язування, які можна написати в рамках даної лабораторної роботи, включає:

- бібліотека, що забезпечує функції роботи з рядками (порівняння, визначення довжини, пошук у рядку послідовності символів тощо);
- бібліотека, що забезпечує функції роботи з однонаправленим або двонаправленим списком;
- бібліотека, що забезпечує функції роботи зі стеком, чергом чи або іншою структурою даних;
- бібліотека, що забезпечує функції роботи з векторами або матрицями (додавання, множення на скаляр, визначення норми тощо).

## 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

### 2.1 Загальна інформація про бібліотеки динамічного зв'язування

#### 2.1.1 Динамічне зв'язування та його відмінності від статичного зв'язування

DLL, як і аплікації, є виконуваними модулями. При цьому між DLL та аплікаціями є певні відмінності:

- на відміну від аплікацій, DLL не можна виконати безпосередньо (вони не є програмами);
- на відміну від аплікацій, у пам'яті одночасно може бути присутня тільки одна копія деякої DLL;
- на відміну від аплікацій, DLL не може мати власного стеку, власної глобальної пам'яті, власної черги повідомлень, не може володіти дескрипторами файлів тощо.

Під час створення аплікацій, після етапу компіляції вихідного коду та створення об'єктних модулів відбувається етап їх зв'язування (linking). Цей етап виконує *редактор зв'язків* (linker), який об'єднує різні об'єктні модулі в єдиний виконуваний файл. Окрім цього, редактор зв'язків формує *таблицю адрес функцій* (function table address), яка містить початкові адреси всіх функцій, які використано в аплікації.

Для створення традиційних аплікацій редактор зв'язків використовує *статичне зв'язування* (static linking), під час якого в аплікацію включають коди всіх функцій всіх статичних бібліотек, які аплікація використовує. На відміну від статичного зв'язування, *динамічне зв'язування* (dynamic linking) дозволяє включати в аплікацію тільки ту інформацію,

яка потрібна для знаходження виконуваного коду функції всередині відповідної бібліотеки динамічного зв'язування.

Іншими словами, використання бібліотек динамічного зв'язування дозволяє різним аплікаціям викликати функції, код яких зосереджено всередині бібліотек, але не є складовою частиною коду самих аплікацій. Функції бібліотек компілюють, зв'язують та зберігають окремо від аплікацій, які їх використовують.

Порівняно з використанням статичного зв'язування, використання динамічного зв'язування має низку переваг:

- процеси, що завантажують одну й ту саму DLL, використовують єдину копію DLL в оперативній пам'яті. Це дозволяє зменшити обсяг використовуваної пам'яті та свопінгу. Натомість, під час статичного зв'язування ОС повинна завантажити в пам'ять окрему копію бібліотеки для кожної аплікації, що її використовує;

- процеси, що використовують одну й ту саму DLL, послуговуються одним і тим самим файлом на диску. У випадку статичного зв'язування код використовуваної бібліотеки включається в кожний виконуваний файл аплікації, що використовує цю бібліотеку;

- коли функції в DLL змінюють, непотрібно повторно компілювати та зв'язувати аплікації, які використовують цю DLL (якщо, звісно, сигнатури функцій не змінено). Під час зміни статично зв'язуваного коду аплікації, які його використовують, потрібно заново компілювати та зв'язувати;

- можна ефективніше здійснювати підтримку аплікацій після їх створення. Наприклад, DLL драйвера дисплею можна з легкістю модифікувати у випадку появи нового дисплею, якого не існувало на момент написання аплікації. При цьому вносити зміни в код самої аплікації непо-

трібно;

- аплікації, написані різними мовами програмування, можуть використовувати одну й ту саму DLL, якщо аплікація та функції DLL слідують однаковому *правилу викликів* (calling convention). Правило викликів визначає порядок запису в стек аргументів функції, яку викликають, хто відповідальний за очищення стеку (функція, яка викликає, чи функція, яку викликають), а також, чи потрібно записувати аргументи функції в регістри процесора;

- полегшення забезпечення інтернаціоналізації аплікації: ресурси, пов'язані з кожною окремою версією, можна помістити в окремі DLL.

Разом із тим, варто звернути увагу, що використання DLL робить аплікацію в певному сенсі залежною від бібліотеки: для належного функціонування аплікації DLL повинна бути присутня в системі.

### 2.1.2 Типи функцій бібліотек динамічного зв'язування

Бібліотеки динамічного зв'язування можуть містити функції двох типів:

- *експортовані* (exported) функції: їх можуть викликати як інші модулі, так і інші функції самої бібліотеки;
- *внутрішні* (internal) функції: їх можуть викликати тільки функції самої бібліотеки.

Для того, щоб процес міг викликати експортовані функції DLL, він спочатку повинен завантажити її та відобразити на свій віртуальний адресний простір. Для кожної окремої DLL ОС підтримує лічильник її зв'язків із процесами. Коли деякий потік завантажує DLL, ОС збільшує лічиль-



ник відповідної DLL на одиницю. Коли потік припиняє роботу з DLL, ОС зменшує цей лічильник на одиницю. Коли процес припиняє роботу або коли значення лічильника для DLL стає рівним 0, ОС вивантажує цю DLL з віртуального адресового простору процесу.

Як і будь-яка інша функція, експортована функція DLL виконується в контексті потоку, який її викликав. У зв'язку з цим справедливі такі твердження:

- потоки процесу, який викликав DLL, можуть використовувати дескриптори, відкриті за допомогою функцій із DLL. Аналогічно, дескриптори, відкриті в будь-яких потоках викликаючого процесу, можна використати всередині функцій із DLL;
- DLL використовує стек викликаючого потоку та віртуальний адресовий простір викликаючого процесу;
- DLL використовує пам'ять у рамках віртуального адресового простору викликаючого процесу.

Окрім функцій, бібліотеки динамічного зв'язування також можуть експортувати дані, але, як правило, дані всередині бібліотеки використовують тільки функції самої бібліотеки.

### 2.1.3 Два типи динамічного зв'язування

Існує два методи динамічного зв'язування, які впливають на спосіб викликання експортованих функцій DLL:

- за *неявного зв'язування* (implicit linking), або *зв'язування етапу завантаження* (load-time linking), виконуваний модуль викликає експортовані функції DLL так, наче це його локальні функції. Для уможливлен-

ня такого викликання, виконуваний модуль на етапі зв'язування потрібно зв'язати з *бібліотекою імпортування* (import library), створеною для DLL, що містить потрібні функції. Бібліотека імпортування надає ОС інформацію, потрібну для завантаження DLL та знаходження експортованих функцій на момент завантаження аплікації;

- за *явного зв'язування* (explicit linking), або *зв'язування етапу виконання* (run-time linking), виконуваний модуль спочатку повинен виконати функцію LoadLibrary (чи LoadLibraryEx) для завантаження DLL під час свого виконання. Виконуваний модуль може викликати експортовані функції, використовуючи *вказівники* на них, отримані за допомогою функції GetProcAddress. За явного зв'язування відпадає потреба у використанні бібліотеки імпортування.

Незважаючи на те, що використання неявного зв'язування дозволяє викликати функції DLL безпосередньо в коді аплікації, а не використовуючи вказівники на них, отримані під час виконання, неявне зв'язування вимагає наявності DLL у системі, навіть якщо аплікація під час свого виконання не викликає жодної функції з DLL. Окрім цього, аплікація на момент компіляції повинна знати імена функцій DLL, які вона буде викликати.

Використання явного зв'язування прибирає ці обмеження за рахунок складнішого способу викликання функції з боку аплікації в момент виконання. Окрім цього, використання явного зв'язування дозволяє забезпечити підтримку додаткових функціональних можливостей, яких не було на момент створення аплікації. Наприклад, якщо аплікація повинна виконувати конвертацію між різними файловими форматами, функції конвертації між конкретними форматами можна винести в окремі DLL. Аплікація може під час виконання аналізувати наявні DLL та використо-

увати відповідні функції конвертації. У випадку появи нових DLL, що забезпечують конвертацію між новими форматами файлів, аплікацію не потрібно створювати наново: вона зможе звертатися до нових функцій за їхніми вказівниками, які вона отримає під час свого виконання.

## 2.2 Створення бібліотеки динамічного зв'язування

### 2.2.1 Складові бібліотеки динамічного зв'язування

Для того, щоб написати будь-яку бібліотеку динамічного зв'язування, потрібно створити:

- один або більше файлів із вихідним кодом. Файли з вихідним кодом можуть містити експортовані функції та дані, внутрішні функції та дані, а також *функцію точки входу* (entry-point function) (див. розділ 2.2.4). Якщо DLL будуть використовувати багатопотокові аплікації, потрібно передбачити синхронізацію доступу до всіх глобальних даних DLL, щоб унеможливити їх пошкодження;

- *файл визначення модуля* (module-definition file) (див. нижче);
- бібліотеку імпортування (тільки для написання бібліотек, які використовують неявне зв'язування). У більшості інструментальних засобів для створення аплікацій Windows, редактор зв'язків створює бібліотеку імпортування DLL автоматично під час створення самої DLL. Створення бібліотеки імпортування з боку програміста виходить за рамки даних методичних рекомендацій.

Структура файлу DLL подібна на структуру файлу .exe, але файл DLL додатково містить *таблицю експортування* (exports table). Ця таблиця містить ім'я кожної експортованої функції DLL. Для того, щоб

функція DLL потрапила в цю таблицю, її потрібно визначити як експортовану. Спосіб визначення функцій DLL як експортованих залежить від інструментальних засобів, які використовують під час створення DLL:

- деякі компілятори дозволяють визначити функції як експортовані за допомогою використання спеціальних *модифікаторів* безпосередньо під час декларування функцій у коді бібліотеки;

- інші компілятори потребують окремого *файлу визначення модуля*, який разом із вихідним кодом подають на редактор зв'язків під час створення бібліотеки.

У наступних розділах розглянемо реалізацію обох цих способів мовою програмування C++.

### 2.2.2 Визначення експортованих функцій за допомогою модифікаторів

Для того, щоб визначити функцію як експортовану безпосередньо у вихідному коді бібліотеки динамічного зв'язування, потрібно використати модифікатор `__declspec(dllexport)` у декларації функції. Цей модифікатор додає в об'єктний файл потрібні директиви експортування, що прибирає потребу у використанні окремого файлу визначень.

Модифікатор `__declspec(dllexport)` потрібно задавати лівіше за ключове слово правила виклику. Наприклад, якщо деяка функція `functionEx` типу `int` з одним аргументом типу `double` використовує правило виклику `WINAPI` (стандартне правило виклику функцій Windows API), то її декларація як експортованої матиме наступний вигляд:

```
__declspec(dllexport) int WINAPI functionEx(double);
```

Модифікатор `__declspec(dllexport)` також можна використувати, щоб визначити як експортовані всі публічні складові деякого класу. Наприклад, для класу `CExample` це виглядатиме наступним чином:

```
class __declspec(dllexport)
    CExampleExport : public CObject
{
    ... class definition ...
};
```

Модифікатор `__declspec(dllexport)` зберігає імена функцій у таблиці експортування DLL.

### Приклад 2.1.

Розгляньмо вихідний код мовою програмування C++ бібліотеки динамічного зв'язування для реалізації найпростіших арифметичних операцій.

```
#include "stdafx.h"
#include <windows.h>

__declspec(dllexport) int WINAPI addNumbers(int num1, int num2)
{
    return num1 + num2;
}

__declspec(dllexport) int WINAPI subtractNumbers(int num1, int num2)
{
    return num1 - num2;
}
```

### 2.2.3 Визначення експортованих функцій за допомогою файлу визначення модуля

Файл визначення модуля — це текстовий файл із розширенням `def`, який містить вирази, що описують атрибути DLL. Використання цього фай-

лу є обов'язковим для компіляції та зв'язування бібліотеки, якщо для визначення експортованих функцій не було використано модифікатор `__declspec(dllexport)` (див. розділ 2.2.2).

У найпростішому випадку, `.def`-файл повинен містити такі вирази:

- вираз `LIBRARY`, який повинен бути першим виразом у файлі. Цей вираз ідентифікує `.def`-файл як належний бібліотеці динамічного зв'язування. За виразом `LIBRARY` повинно слідувати ім'я `DLL`. Редактор зв'язків записує це ім'я в бібліотеку імпорту `DLL`;

- вираз `LIBRARY`, який містить перелік імен (і, можливо, порядкові номери) експортованих функцій `DLL`. Порядковим номерам функцій повинен передувати символ `@`. Порядкові номер повинні належати проміжку від 1 до  $N$ , де  $N$  — загальна кількість експортованих функцій `DLL`.

### Приклад 2.2.

Розгляньмо текст файлу визначення модуля для бібліотеки динамічного зв'язування з Прикладу 2.1.

```
LIBRARY DllAddSubtract

EXPORTS
addNumbers @1
subtractNumbers @2
```

#### 2.2.4 Функція точки входу

У бібліотеці динамічного зв'язування може існувати необов'язкова *функція точки входу*. ОС викликає цю функцію кожного разу, коли потік завантажує чи вивантажує `DLL`, тому її можна використовувати для виконання ініціалізації або очищення пам'яті.

ОС викликає функцію точки входу в контексті того процесу (поток), який завантажив чи вивантажив `DLL`. Це дозволяє `DLL` використовувати функцію точки входу для виділення пам'яті у віртуальному адресовому просторі викликаючого процесу чи для відкриття дескрипторів, доступних

викликаючому процесові. В один і той же момент часу тільки один потік може викликати функцію точки входу.

Розгляньмо детальніше особливості функції точки входу. Ця функція має наступний формат:

```
BOOL WINAPI DllMain(
    HINSTANCE hinstDLL,
    DWORD fdwReason,
    LPVOID lpvReserved
);
```

Назва функції необов'язково повинна бути DllMain, програміст за потреби може назвати її інакше. Аргумент `hinstDLL` цієї функції відповідає за дескриптор DLL (фактично, базову адресу DLL), аргумент `fdwReason` визначає причину, чому функцію було викликано:

- якщо `fdwReason` дорівнює `DLL_PROCESS_ATTACH`, то DLL завантажено у віртуальний адресовий простір процесу. У цьому разі функція може виконати ініціалізацію потрібних структур даних процесу;

- якщо `fdwReason` дорівнює `DLL_PROCESS_DETACH`, то DLL вивантажено з віртуального адресового простору процесу. У цьому разі функція може виконати очищення пам'яті від потрібних структур даних у рамках відповідного процесу;

- якщо `fdwReason` дорівнює `DLL_THREAD_ATTACH`, то поточний процес створює новий потік. У цьому разі ОС викликає функції точок входу всіх DLL, які завантажено в адресовий простір процесу, при цьому виклики здійснюються у контексті новостворюваного потоку. У цьому разі функція може виконати ініціалізацію потрібних структур даних потоку;

- якщо `fdwReason` дорівнює `DLL_THREAD_DETACH`, то потік завершує своє виконання в штатному режимі. У цьому випадку функція може виконати очищення пам'яті від потрібних структур даних у рамках відповідного потоку.

Аргумент `lpvReserved` функції точки входу дорівнює:

- якщо `fdwReason` дорівнює `DLL_PROCESS_ATTACH` — `NULL` у випадку явного зв'язування та будь-якому іншому значенню у випадку неяв-

ного зв'язування;

- якщо `fdwReason` дорівнює `DLL_PROCESS_DETACH` — `NULL` у випадку виклику функції `FreeLibrary` (див. розділ 2.3.2) чи неуспішного завантаження DLL та будь-якому іншому значенню у випадку завершення роботи процесу.

Якщо `fdwReason` дорівнює `DLL_PROCESS_ATTACH`, функція точки входу повертає `TRUE` у випадку успішного завершення та `FALSE` у випадку помилки. Для всіх інших значень `fdwReason` значення, яке повертає функція точки входу, можна проігнорувати.

Функцію точки входу повинно бути декларовано з правилом виклику `WINAPI`. В іншому випадку, DLL завантажено не буде, і ОС виведе на екран повідомлення про відповідну помилку.

Деталі використання функції точки входу виходять за рамки даних методичних рекомендацій.

## 2.3 Використання бібліотек динамічного зв'язування в аплікаціях

У даному розділі розглянемо способи використання бібліотек динамічного зв'язування в аплікаціях, написаних мовою C++ із використанням Windows API.

### 2.3.1 Використання бібліотек неявного динамічного зв'язування

Для того, щоб використати в аплікації бібліотеку неявного динамічного зв'язування, потрібно:

- у файлах заголовків (header file, .h-file) відповідної аплікації зазначити декларації експортованих функцій, класів та/чи даних відповідної DLL. Експортовані класи, функції та/чи дані повинно бути декларовано з моди-



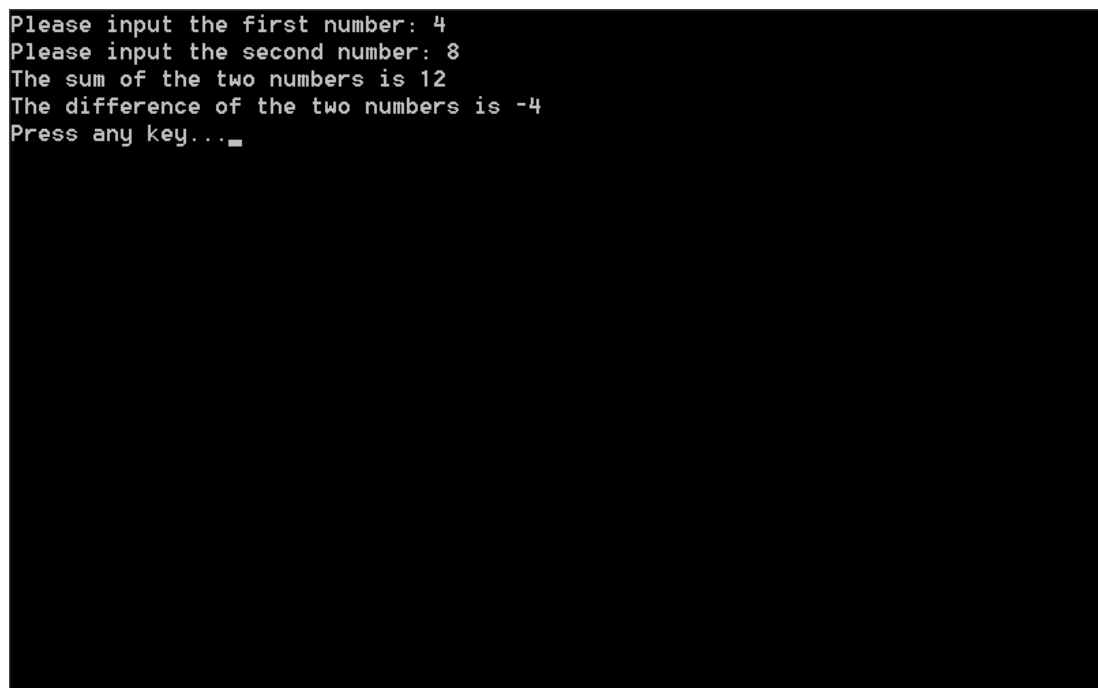
фікатором `__declspec(dllimport);`

- скомпілювати вихідні файли аплікації та зв'язати їх із бібліотекою імпорту відповідної DLL.

За виконання цих умов у вихідному коді аплікації експортовані функції DLL можна викликати як локальні функції.

### Приклад 2.3.

Розгляньмо вихідний текст програми на мові програмування C++, яка демонструє використання бібліотеки динамічного зв'язування з Прикладу 2.1 у режимі неявного зв'язування (рисунок 2.1).



```
Please input the first number: 4
Please input the second number: 8
The sum of the two numbers is 12
The difference of the two numbers is -4
Press any key..._
```

Рисунок 2.1 – Результат виконання програми з Прикладу 2.3

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>

using namespace std;

//декларації експортованих функцій DLL
__declspec(dllimport) int WINAPI addNumbers(int num1, int num2);
__declspec(dllimport) int WINAPI subtractNumbers(int num1, int num2);

int _tmain(int argc, _TCHAR* argv[])
{
```

```

int num1;
int num2;

//запитуємо в користувача два числа
cout << "Please input the first number: ";
cin >> num1;

cout << "Please input the second number: ";
cin >> num2;

//виводимо на екран результати застосування наших функцій
cout << "The sum of the two numbers is " <<
    addNumbers(num1, num2) << endl;
cout << "The difference of the two numbers is "
    << subtractNumbers(num1, num2) << endl;

cout << "Press any key...";
getchar();
getchar();

return 0;
}

```

Розгляньмо послідовність дій, які виконує ОС під час завантаження в пам'ять аплікації, яка використовує бібліотеки неявного динамічного зв'язування:

а) ОС використовує інформацію з завантажувального модуля, яку туди помістив редактор зв'язків, для знаходження імен використовуваних бібліотек;

б) ОС намагається знайти ідентифіковані бібліотеки. Порядок пошуку бібліотек в ОС Windows наступний:

- 1) каталог, у якому знаходиться виконуваний модуль аплікації;
- 2) поточний каталог;
- 3) системний каталог Windows;
- 4) каталог Windows;
- 5) каталоги, перелічені у змінній середовища PATH;

якщо ОС не може знайти потрібні бібліотеки, вона перериває виконання процесу та виводить на екран відповідне повідомлення про помилку;

в) у разі успішного знаходження всіх бібліотек, ОС відображає їх на віртуальний адресовий простір процесу аплікації та збільшує на 1 лічильник зв'язків із процесами кожної бібліотеки;

г) ОС викликає функції точки входу кожної бібліотеки (якщо вони

є). Якщо якась із цих функцій не повертає TRUE, ОС перериває виконання процесу та виводить на екран відповідне повідомлення про помилку;

д) ОС додає в таблицю адрес функцій початкові адреси функцій, імпортованих із відповідних бібліотек.

Варто зауважити, що під час ініціалізації процесу аплікації DLL не завантажують, а тільки відображають на відповідний віртуальний адресовий простір. Завантаження DLL у пам'ять відбувається тільки у випадку виклику функцій DLL під час виконання аплікації.

### 2.3.2 Використання бібліотек явного динамічного зв'язування

Для того, щоб викликати функції з бібліотеки динамічного зв'язування з використанням явного зв'язування, аплікація повинна спочатку завантажити відповідну бібліотеку за допомогою функції `LoadLibrary`. Формат цієї функції наступний:

```
HMODULE WINAPI LoadLibrary(LPCTSTR lpFileName);
```

Аргумент `lpFileName` цієї функції визначає ім'я бібліотеки (назву відповідного .dll-файлу). ОС намагається знайти відповідну бібліотеку, послідовно переглядаючи каталоги в порядку, вказаному в розділі 2.3.1. Якщо ОС не знаходить відповідну бібліотеку в жодному з цих каталогів, функція повертає NULL. Функція також повертає NULL, якщо функція точки входу повертає FALSE (див. розділ 2.2.4).

Якщо ОС знаходить потрібну бібліотеку, вона відображає бібліотеку на адресовий простір процесу аплікації, збільшує лічильник зв'язків із процесами даної бібліотеки на одиницю та повертає дескриптор модуля бібліотеки. Якщо на вхід `LoadLibrary` подано бібліотеку, яку вже відображено на адресовий простір процесу, то функція просто повертає дескриптор модуля бібліотеки і збільшує лічильник зв'язків із процесами даної бібліотеки на одиницю.

Якщо бібліотека містить функцію точки входу, ОС викликає її в контексті потоку, який викликав `LoadLibrary`. ОС не викликати функцію точки входу, якщо бібліотеку вже раніше було завантажено за допомогою функції `LoadLibrary`, але не було вивантажено за допомогою функції `FreeLibrary` (див. нижче).

Після завантаження бібліотеки аплікація повинна отримати вказівник на функцію цієї бібліотеки, яку вона хоче викликати. Для цього аплікація може викликати функцію `GetProcAddress`. Формат цієї функції наступний:

```
FARPROC WINAPI GetProcAddress(
    HMODULE hModule,
    LPCSTR lpProcName
);
```

Аргумент `hModule` цієї функції визначає дескриптор модуля DLL, яка містить шукану функцію для виклику, аргумент `lpProcName` — ім'я шуканої функції. У випадку успішного завершення, функція повертає адресу шуканої функції. У випадку помилки функція повертає `NULL`.

Під час використання вказівника на функцію з бібліотеки потрібно акуратно вказувати аргументи цієї функції. Оскільки для виклику функції за її вказівником компілятор не виконує перевірку на відповідність типів аргументів, відповідальність за їх коректне визначення лягає на програміста. Для спрощення контролю типів можна використати директиву `typedef` для визначення нового типу вказівника на відповідну функцію, зазначивши типи її аргументів (див. Приклад 2.4).

Якщо аплікація більше не потребує використання функцій бібліотеки, вона може викликати функцію `FreeLibrary` для вивантаження бібліотеки з пам'яті. Формат цієї функції наступний:

```
BOOL WINAPI FreeLibrary(HMODULE hModule);
```

Аргумент `hModule` цієї функції визначає дескриптор модуля DLL, яку потрібно вивантажити. У випадку успішного завершення, функція повертає ненульове значення та зменшує значення лічильника зв'язків із процесами відповідної DLL на одиницю. Якщо лічильник стає рівним 0, то

функція вивантажує DLL із пам'яті. У випадку помилки функція повертає 0.

### Приклад 2.4.

Розгляньмо вихідний текст програми на мові програмування C++, яка демонструє використання бібліотеки динамічного зв'язування з Прикладу 2.1 у режимі неявного зв'язування.

```
#include "stdafx.h"
#include <windows.h>
#include <iostream>

using namespace std;

//нові корисні типи для вказівників на функції
typedef INT (WINAPI* ADDNUMBER)(INT, INT);
typedef INT (WINAPI* SUBTRACTNUMBER)(INT, INT);

int _tmain(int argc, _TCHAR* argv[])
{
    HMODULE hModule;
    ADDNUMBER lpAddNumber;
    SUBTRACTNUMBER lpSubtractNumber;
    int num1, num2;

    //запитуємо в користувача два числа
    cout << "Please input the first number: ";
    cin >> num1;

    cout << "Please input the second number: ";
    cin >> num2;

    //завантажуємо потрібну DLL
    hModule = LoadLibrary(L"DllAddSubtract.dll");

    if(hModule != NULL)
    {
        //отримуємо вказівники на потрібні функції
        lpAddNumber = (ADDNUMBER) GetProcAddress(
            hModule, "addNumbers");

        if(!lpAddNumber)
        {
            //якщо сталася помилка
            cout << "Error while loading function addNumbers!";

            //вивантажуємо бібліотеку
            FreeLibrary(hModule);
            return -1;
        }
        else
        {
            lpSubtractNumber = (SUBTRACTNUMBER) GetProcAddress(
```

```

        hModule, "subtractNumbers");

    if(!lpSubtractNumber)
    {
        //якщо сталася помилка
        cout << "Error while loading function subtractNumbers!";

        //вивантажуємо бібліотеку
        FreeLibrary(hModule);
        return -1;
    }
    else
    {
        //викликаємо наші функції
        cout << "The sum of the two numbers is " <<
            lpAddNumber(num1, num2) << endl;
        cout << "The difference of the two numbers is " <<
            lpSubtractNumber(num1, num2) << endl;

        cout << "Press any key...";
        getchar();
        getchar();

        //вивантажуємо бібліотеку
        FreeLibrary(hModule);
    }
}
else
{
    cout << "Error while loading the DLL!";
}

return 0;
}

```

### 3 ПОРЯДОК ЗДАЧІ ЛАБОРАТОРНОЇ РОБОТИ ТА ВИМОГИ ДО ЗВІТУ

Здача лабораторної роботи передбачає:

- демонстрацію працездатності розробленої в рамках роботи програми (із використанням як неявного, так і явного зв'язування написаної бібліотеки) в ОС Windows версії, не нижчої за 2000;
- здачу викладачеві звіту з лабораторної роботи.

Під час здавання лабораторної роботи викладач має право ставити студентові питання за матеріалами розділу 2.

Під час демонстрації роботи програми викладач має право вимагати від студента внесення змін як у бібліотеку, так і в програму, що її використовує.

Звіт про виконання лабораторної роботи повинен включати:

- а) вступ;
- б) постановку задачі;
- в) теоретичні відомості;
- г) опис розробленої програми;
- д) результати випробування розробленої програми;
- е) висновки;
- є) посилання на літературні джерела;
- ж) додаток, у якому повинно бути розміщено лістинг розробленої програми.

Детальніші вимоги до звіту викладено в документі «Вимоги до оформлення звіту». Окрім вимог, викладених у ньому, у розділі «Опис розробленої програми» потрібно додатково подати опис функцій, які визначено в бібліотеці динамічного зв'язування.

Текстову частину повинно бути оформлено відповідно до вимог ДСТУ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення» із застосуванням системи підготовки документів L<sup>A</sup>T<sub>E</sub>X.

## ПЕРЕЛІК ПОСИЛАНЬ

1. Microsoft Library [Електронний ресурс]. — Режим доступу: <https://msdn.microsoft.com/en-us/library/>
2. Petzold C. Programming Windows / C. Petzold. — [5th ed.] — Microsoft Press, 1998. — 1100 p.
3. Simon R. Windows NT Win32 API SuperBible / R. Simon. — Waite Group Press, 1997. — 1510 p.