# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**ЗВІТ**

**Про виконання лабораторної роботи №5**

**«Робота з динамічними бібліотеками в ОС Windows»**

**з дисципліни «Операційні системи»**

Виконав:

слухач групи ПЗС-11

Гринчук Тарас

Прийняв:

доц. Яковина В.С.

« »\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2014 р.

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛЬВІВ – 2014

**Тема роботи**: Робота з динамічними бібліотеками в ОС Windows.

**Мета роботи**: Ознайомитися з динамічно-зв’язувальними бібліотеками (Dynamic-link library) в ОС Windows. Навчитися реалізовувати динамічно-зв’язувальні бібліотеки.

## 1. Теоретичні відомості

Динамічні бібліотеки (dynamic - link libraries, DLL) - наріжний камінь операційної системи Windows, починаючи з найпершої її версії. У DLL містяться всі функції Windows API. Три найважливіші DLL: Kernel32.dll (управління пам'яттю, процесами і потоками), User32.dll (підтримка для користувацького інтерфейсу, у тому числі функції, пов'язаних із створенням вікон і передачею повідомлень) і GDI32.dll (графіка і вивід тексту).

У Windows є й інші DLL, функції яких призначені для більш спеціалізованих завдань. Наприклад, в AdvAPI32.dll містяться функції для захисту об'єктів, роботи з реєстром і реєстрації подій, в ComDlg32.dll - стандартні діалогові вікна (на кшталт File Open і File Save), a ComCrl32.dll підтримує стандартні елементи управління.

Основні з причин, по яких потрібно застосовувати DLL:

* *Розширення функціональності програми*. DLL можна завантажувати в адресний простір процесу динамічно, що дозволяє додатку, визначивши, які дії від нього потрібні, довантажувати потрібний код, тому одна компанія, створивши якийсь додаток, може передбачити розширення його функціональності за рахунок DLL від інших компаній.
* *Можливість використання різних мов програмування.* У Вас є вибір, якою мовою писати ту чи іншу частину додатку. Так, користувальницький інтерфейс програми Ви швидше за все будете створювати на Microsoft Visual Basic, але прикладну логіку найкраще реалізувати на С ++ . Програма на Visual Basic може завантажувати DLL, написані на С ++, Коболі, Фортрані та ін.
* *Більш просте управління проектом.* Якщо в процесі розробки програмного продукту окремі його модулі створюються різними групами, то при використанні DLL таким проектом управляти набагато простіше. Проте кінцева версія програми повинна включати якомога менше файлів.
* *Економія пам'яті.* Якщо одну і ту ж DLL використовує кілька додатків, в оперативній пам'яті може зберігатися тільки один її примірник, доступний цим програмам. Приклад - DLL-версія бібліотеки С / С++. Нею користуються багато додатків. Якщо всіх їх скомпонувати із статично підключеною версією цієї бібліотеки, то код таких функцій, як sprintf, strcpy, malloc та ін, буде багаторазово дублюватися в пам'яті. Але якщо вони компонуються з DLL -версією бібліотеки С / С++, в пам'яті буде присутня лише одна копія коду цих функцій, що дозволить набагато ефективніше використовувати оперативну пам'ять.
* *Поділ ресурсів.* DLL можуть містити такі ресурси, як шаблони діалогових вікон, рядки, значки і бітові карти (растрові зображення). Ці ресурси доступні будь-яким програмам.
* *Спрощення локалізації.* DLL нерідко застосовуються для локалізації додатків. Наприклад, додаток, що містить тільки код без всяких компонентів для користувацького інтерфейсу, може завантажувати DLL з компонентами локалізованого інтерфейсу.
* *Рішення проблем, пов'язаних з особливостями різних платформ.* У різних версіях Windows містяться різні набори функцій. Найчастіше розробникам потрібні нові функції, що існують в тій версії системи, якою вони користуються. Якщо Ваша версія Windows не підтримує ці функції, Вам не вдасться запустити такий додаток: завантажувач просто відмовиться його запускати. Але якщо ці функції будуть знаходитися в окремій DLL, Ви завантажите програму навіть у більш ранніх версіях Windows, хоча скористатися ними Ви все одно не зможете.
* *Реалізація специфічних можливостей.* Певна функціональність в Windows доступна тільки при використанні DLL. Наприклад, окремі види пасток (що встановлюються викликом SetWindowsHookEx і SetWinEventHook можна задіяти за тієї умови, що функція повідомлення пастки розміщена в DLL. Крім того, розширення функціональності оболонки Windows можливо лише за рахунок створення СОМ-об'єктів, існування яких допустимо тільки в DLL. Це ж відноситься і до завантажуваних Web-браузером ActiveX-елементів, що дозволяє створювати Web-сторінки з більш багатою функціональністю.

Збираючи виконуваний модуль, який імпортує функції і змінні з DLL, Ви повинні спочатку створити цю DLL. А для цього потрібно наступне:

1. Перш за все Ви повинні підготувати заголовний файл з прототипами функцій, структурами і ідентифікаторами, експортованими з DLL. Цей файл включається у вихідний код всіх модулів Вашої DLL. Як Ви потім побачите, цей же файл знадобиться і при складанні виконуваного модуля (або модулів), який використовує функції і змінні з Вашої DLL.
2. Ви пишете на С/С ++ модуль (або модулі) вихідного коду з тілами функцій і визначеннями змінних, які повинні знаходитися в DLL. Так як ці модулі вихідного коду не потрібні для збірки виконуваного модуля, вони можуть залишитися комерційною таємницею компанії-розробника.
3. Компілятор перетворює вихідний код модулів DLL в OBJ-файли (по одному на кожен модуль).
4. Компоновщик збирає всі OBJ-модулі в єдиний завантажувальний DLL-модуль, в який зрештою поміщаються бінарний код і змінні (глобальні та статичні), що відносяться до даної DLL. Цей файл буде потрібно при компіляції виконуваного модуля.
5. Якщо компонувальник виявить, що DLL експортує хоча б одну змінну або функцію, то створить і LIB – файл. Цей файл зовсім крихітний, оскільки в ньому немає нічого, крім списку символьних імен функцій і змінних, що експортуються з DLL. Цей LIB-файл теж знадобиться при компіляції ЕХЕ-файла.
6. Створивши DLL , можна перейти до складання виконуваного модуля. У всі модулі вихідного коду, де є посилання на зовнішні функції, змінні, структури даних або ідентифікатори, треба включити заголовний файл, наданий розробником DLL.

Створивши DLL і ЕХЕ-модулі, додаток можна запустити. При його запуску завантажувач операційної системи виконує такі операції:

1. Завантажувач операційної системи створює віртуальний адресний простір для нового процесу і проектує па нього виконуваний модуль.
2. Далі завантажувач аналізує розділ імпорту, знаходить всі необхідні DLL-модулі і теж проектує на адресний простір процесу. Зауважте, що DLL може імпортувати функції та змінні іншої DLL, а значить, у неї може бути власний розділ імпорту. Закінчуючи підготовку процесу до роботи, завантажувач переглядає розділ імпорту кожного модуля і проектує всі необхідні DLL-модулі на адресний простір цього процесу. Як бачите, на ініціалізацію процесу може піти досить тривалий час.

## 2. Хід роботи

## Завдання (варіант №5).

1. Реалізувати лабораторну роботу №4 у вигляді динамічно-зв’язувальної бібліотеки: створити масив N елементів і відсортувати його елементи у порядку зростання за допомогою методу «вибірки» (елементи масиву згенерувати випадковим чином за допомогою вбудованих функцій). Виконати розпаралелювання заданого алгоритму на 2, 4, 8 потоки із використанням синхронізації та реалізувати прогрес виконання задачі.
2. Запустити створену бібліотеку з командної стрічки за допомогою rundll32.exe.
3. Створити окрему програму і реалізувати статичний зв'язок між програмою та бібліотекою.
4. Створити окрему програму і реалізувати динамічний зв'язок між програмою та бібліотекою
5. Експортувати головну функцію бібліотеки під іншим іменем.
6. Результати виконання роботи відобразити у звіті.
7. Створення проекту бібліотеки динамічного компонування (DLL) у ***Microsoft VC++ 2010***, складається з таких кроків:
   1. У меню **Файл** обираємо пункт **Створити** і потім пункт **Проект ...**.
   2. У вузлі **Visual C++** області **Типи проектів** вибераємо **Win32**.
   3. В області **Шаблони** вибераємо **Консольний додаток Win32**.
   4. Виберіть ім'я проекту: **MathFuncsDll**, і введемо його в поле **Ім'я**. Виберемо ім'я рішення: **DynamicLibrary**, і введемо його в полі **Ім'я рішення**.
   5. Для запуску **майстра додатків Win32** натиснаємо кнопку **ОК**. На сторінці **Загальні відомості** діалогового вікна **Майстер додатків Win32** натискаємо кнопку **Далі**.
   6. На сторінці **Параметри** додатка діалогового вікна **Майстер додатків Win32**, в поле **Тип програми**, вибераємо пункт **DLL.**
   7. На сторінці **Параметри додатка** діалогового вікна **Майстер додатків Win32** в поле **Додаткові параметри** вибераємо пункт **Порожній проект**.
   8. Щоб створити проект, натискаємо кнопку **Готово**.

Додавання функцій в бібліотеку динамічного компонування:

1. Щоб створити файл заголовка для нашого проекту, в меню **Проект** вибераємо команду **Додати новий елемент...**. Відкриється діалогове вікно **Додавання нового елемента**. У вузлі **Visual C++** області **Категорії** вибераємо пункт **Код**. В області **Шаблони** вибираємо пункт **Заголовний файл (.h)**. Вибираємо ім'я файлу заголовка: **MathFuncsDll.h**, і натискаємо кнопку **Додати**. Відобразиться порожній файл.
2. Додамо наступний код:

*#include <windows.h>*

*namespace MathFuncs {*

*\_\_declspec(dllexport) int mainDLL();*

*}*

1. Звернемо увагу на модифікатор *\_\_declspec (dllexport)* в оголошеннях методів у цьому коді. Цей модифікатор дозволяє експорт методу бібліотекою DLL для використання його іншими додатками.
2. Щоб створити вихідний файл для бібліотеки, в меню **Проект** вибираємо команду **Додати новий елемент ...**. Відкриється діалогове вікно **Додавання нового елемента**. У вузлі **Visual C++** області **Категорії** вибираємо пункт **Код**. В області **Шаблони** вибираємо пункт **Файл C++ (.сpp)**. Вибираємо ім'я початкового файлу: **MathFuncsDll.cpp**, і натискає кнопку **Додати**. Відобразиться порожній файл.
3. Реалізуємо функціональність класу бібліотеки у вихідному файлі. Код повинен виглядати приблизно таким чином:

*#include "MathFuncsDll.h"*

*... ... ... ... ... ...*

*namespace MathFuncs {*

*int mainDLL() {*

*... ... ... ... ... ...*

*}*

*}*

1. Щоб побудувати бібліотеку DLL проекту, в меню **Проект** виберемо **Властивості MathFuncsDll**. У лівій області в полі **Властивості конфігурації** виберемо **Загальні**. У правій області в полі **Тип конфігурації** вибираємо **Динамічна бібліотека (.Dll)**. Натискаємо кнопку **ОК** для збереження змін.
2. Скомпілюємо бібліотеку динамічного компонування, вибравши команду **Побудувати рішення** в меню **Побудова**. У результаті буде створена бібліотека DLL, яка може використовуватися іншими програмами.
3. Для запуску бібліотеки з командної стрічки за допомогою rundll32.exe, вона повинна мати експортну функцію (точку входу), з прототипом, оформленим в наступному порядку:

*extern "C" \_\_declspec(dllexport) void CALLBACK OutputFile(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow);*

Сама ж реалізація функції *OutputFile*, має вигляд:

*void CALLBACK OutputFile(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow){*

*toScreen = 0;*

*main();*

*}*

Глобальна змінна toScreen вказує на те, що вивід будемо здійснювати у файл, оскільки при роботі rundll32.exe, консоль працює не коректно. Загалом код бібліотеки слід доопрацювати наступним чином:

*#include "MathFuncsDll.h"*

*int toScreen = 0; //1 - вивід на екран, інакше - у файл*

*FILE \*A; //файл для виводу результатів*

*... ... ... ... ... ...*

*namespace MathFuncs {*

*//для виклику з rundll32*

*void CALLBACK OutputFile(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow){*

*toScreen = 0;*

*main();*

*}*

*}*

*}*

Також потрібно всюди у коді реалізувати вивід у файл чи екран, в залежності від змінної *toScreen*, так, наприклад виглядатиме функція *progress*:

*//функція виводу прогресу на екран*

*void progress(int value) {*

*if(toScreen) {*

*system("cls");*

*printf("Сортування... %d %%\n\n", value);*

*} else*

*fprintf(A, "Сортування... %d %%\n\n", value);*

*for(int i = 0; i < threadFinalsIndex; i++)*

*if(toScreen) printf("Потiк %d завершено. /К-сть елементiв = %d/\n", threadFinals[i]+1, size[threadFinals[i]]);*

*else fprintf(A, "Потiк %d завершено. /К-сть елементiв = %d/\n", threadFinals[i]+1, size[threadFinals[i]]);*

*}*

Крім цього, слід зауважити, що компілятор змінює назву фунції, додаючи службові дані. Скористаємося утилітою *DLL Export viewer*, для отримання кінцевої назви. Як бачимо на рис. 2.1, точка входу набуде назви: «\_*OutputFile@16*».

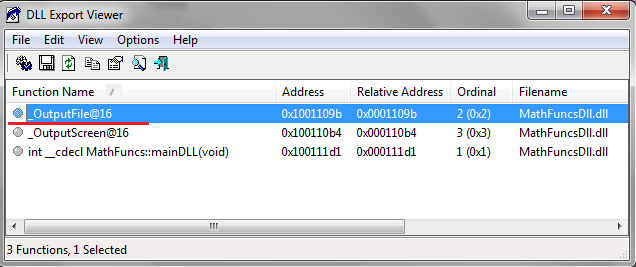


Рис. 2.1. Утиліта *DLL Export viewer*

Для зручності запуску, в каталозі з нашою DLL, створимо bat-файл, що запускатиме бібліотеку в rundll32.exe, а потім відкриватиме результуючий файл у блокноті. Ось текст цього bat-файлу:

*rundll32 MathFuncsDll.dll \_OutputFile@16*

*notepad.exe d:\output.txt*

1. Створимо програму, що посилається на бібліотеку динамічного компонування через статичний зв’язок:
   1. В меню Файл виберемо пункт **Створити** і потім пункт **Проект ...**.
   2. У вузлі **Visual C++** області **Типи проектів** виберемо **Win32**.
   3. В області **Шаблони** виберемо **Консольний додаток Win32**.
   4. Виберемо ім'я проекту: **MyExecRefsDll**, і введемо його в поле **Ім'я**.
   5. У списку поруч із полем **Рішення** виберемо пункт **Додати в рішення**. Після цього новий проект буде додано до того ж рішення, що і бібліотека динамічного компонування.
   6. Для запуску майстра додатків Win32 натиснемо кнопку **ОК**. На сторінці **Загальні відомості** діалогового вікна **Майстер додатків Win32** натиснемо кнопку **Далі**.
   7. На сторінці **Параметри додатка** діалогового вікна **Майстер додатків Win32** в поле **Тип програми** виберемо пункт **Консольний додаток**.
   8. На сторінці **Параметри додатка** діалогового вікна **Майстер додатків Win32** в поле **Додаткові параметри** знімимо прапорець **Передкомпільований заголовок**.
   9. Щоб створити проект , натиснимо кнопку **Готово**.
   10. По завершенні створення консольного додатка буде створена порожня програма. Ім'я вихідного файлу буде збігатися з ім'ям, вибраним раніше для проекту. У нас він має ім'я **MyExecRefsDll.cpp**.
   11. Для використання функцій з бібліотеки динамічного компонування необхідно послатися на цю бібліотеку. Для цього в оглядачі рішень виберемо проект **MyExecRefsDll** і потім виберемо **Посилання ...** в меню **Проект**. У діалоговому вікні **Сторінки властивостей** розгорнемо вузол **Загальні властивості**, виберемо пункт **.NET Framework** **і посилання** і натиснемо кнопку **Додати нове посилання...**.
   12. З'явиться діалогове вікно **Додавання посилання**. У цьому діалоговому вікні відображається список всіх бібліотек, на які можна посилатися. На вкладці **Проект** перераховуються всі проекти поточного рішення і включені в них бібліотеки. На вкладці **Проекти** виберемо **MathFuncsDll**. Потім натиснемо кнопку **ОК**.
   13. Для створення посилання на заголовні файли бібліотеки динамічного компонування необхідно змінити шлях до каталогів включення. Для цього в діалоговому вікні **Вікна властивостей** послідовно розгорнемо вузли **Властивості конфігурації, C/C++**, а потім виберемо **Загальні**. Поруч з полем **Додаткові каталоги включення** введемо шлях до місця розміщення файлу заголовків **MathFuncsDll.h**.
   14. Виконуваний файл не завантажує бібліотеки динамічного компонування аж до часу виконання. Необхідно вказати системі місце для пошуку бібліотеки **MathFuncsDll.dll**. Це можна зробити за допомогою змінної середовища **PATH**. Для цього в діалоговому вікні **Вікна властивостей** розгорнемо вузол **Властивості конфігурації**, а потім виберемо **Налагодження**. У полі введемо наступний рядок: *PATH = <шлях\_до\_файлу\_MathFuncsDll.dll>.* Натиснемо кнопку **ОК** для збереження всіх змін .
   15. Тепер клас **MyMathFuncs** можна використовувати в додатку. Замінимо код у файлі **MyExecRefsDll.cpp** наступним кодом:

#include "MathFuncsDll.h"

using namespace MathFuncs;

int main() {

//викликаємо функцію mainDLL з MathFuncsDll.DLL

return mainDLL();

}

1. Створимо програму, що посилається на бібліотеку динамічного компонування через динамічний зв’язок:
   1. Нам потрібно у DLL мати експортну процедуру, яка буде запускати роботу бібліотеки з нашої програми. Розробимо її по аналогії до випадку, коли нам потрібно було запускати DLL з rundll32.exe, за винятком, що дані виводимуться на екран. Ось її прототип:

*extern "C" \_\_declspec(dllexport) void CALLBACK OutputScreen(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow);*

І, власне, реалізація:

*//для виклику, при пізньому зв'язуванні*

*void CALLBACK OutputScreen(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow){*

*toScreen = 1;*

*main();*

*}*

* 1. Створимо новий проект ***DynamicLinking***. Текст програми ***DynamicLinking.cpp***:

#include <windows.h>

#include <stdio.h>

#include <conio.h>

#include <iostream>

using namespace std;

//тип вказівника на функцію

typedef int (\_\_cdecl \*MYPROC)(LPVOID);

int main() {

HINSTANCE hinstLib;

MYPROC ProcFromDLL; //вказівник на нашу функцію

//отримуємо дескриптор нашої dll

hinstLib = LoadLibrary(TEXT("MathFuncsDll.dll"));

if (hinstLib != NULL) {

//отримуємо адресу імпортованої функції

ProcFromDLL = (MYPROC) GetProcAddress(hinstLib, "\_OutputScreen@16");

//виклик функції

if (NULL != ProcFromDLL) (ProcFromDLL)((LPVOID)0);

//очистимо пам'ять після dll

FreeLibrary(hinstLib);

} else {

setlocale(LC\_ALL,"Ukrainian");

printf("Помилка завантаження MathFuncsDll.dll\n");

}

return 0;

}

* 1. Файл **MathFuncsDll.dll** потрібно скопіювати у папку з проектом ***DynamicLinking***, інакше на екрані з’явиться відповідне повідомлення.

1. В залежності від випадку, ми експортуємо головну функцію бібліотеки під іменами: mainDLL, \_OutputFile@16, \_OutputScreen@16.
2. Кінцевий текст програмних модулів динамічної бібліотеки міститься в розділі 3, а результати виконання див. розділ 4.

## Текст програмних модулів DLL

//MathFuncsDLL.h

#include <windows.h>

namespace MathFuncs {

//прототипи функцій, що експортуються

\_\_declspec(dllexport) int mainDLL();

extern "C" \_\_declspec(dllexport) void CALLBACK OutputFile(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow);

extern "C" \_\_declspec(dllexport) void CALLBACK OutputScreen(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow);

}

//MathFuncsDLL.cpp

#include "MathFuncsDll.h"

#include <stdexcept>

#include <iostream>

#include <conio.h>

#include <windows.h>

#include <stdio.h>

#include <cstdlib>

#include <string>

#include <math.h>

#include <time.h>

using namespace std;

//==============================================================

int toScreen = 0; //1 - вивід на екран, інакше - у файл

FILE \*A; //файл для виводу результатів

//глобальні константи

const int N = 30; //к-сть елементів масиву

const int K = 8; //к-сть потоків

int a[N]; //вхідний масив

int b[K][N]; //вихідний масив розбитий на потоки

int size[K]; //к-сть елементів в рядках масиву b[K][N]

//масив порядку завершення потоків

int threadFinals[K];

//індекс цього масиву

LONG threadFinalsIndex = 0;

//загальна к-сть ітерацій алгоритму сортування

long itersNum;

//поточна к-сть ітерацій, яка вже виконана програмою

LONG itersCur = 0;

//"прапорець" для спін-блокування

LONG resourceInUse = FALSE;

//критичні секції

CRITICAL\_SECTION cs, cs2;

//дескриптор семафора

HANDLE hSemaphore;

//дескриптор мютекса

HANDLE hMutex;

//==============================================================

//функція виводу прогресу на екран

void progress(int value) {

if(toScreen) {

system("cls");

printf("Сортування... %d %%\n\n", value);

} else

fprintf(A, "Сортування... %d %%\n\n", value);

for(int i = 0; i < threadFinalsIndex; i++)

if(toScreen) printf("Потiк %d завершено. /К-сть елементiв = %d/\n", threadFinals[i]+1, size[threadFinals[i]]);

else fprintf(A, "Потiк %d завершено. /К-сть елементiв = %d/\n", threadFinals[i]+1, size[threadFinals[i]]);

}

//==============================================================

//функція сортування

void sort(int arr[], int n, int threadNum) {

for (int i = 0; i < n; i++) {

// Знайдемо мінімальний елемент на

// проміжку індексів [i; n);

// спочатку його індекс рівний i

int minValueIndex = i;

// Перебираємо елементи, що залишилися на проміжку

for (int j = i + 1; j < n; j++) {

// Якщо елемент в позиції j менший

// елемента в позиції minValueIndex, то

// необхідно обновити значення індекса

if (arr[j] < arr[minValueIndex]) {

minValueIndex = j;

}

//Спін-блокування:

//чекаємо доступу до ресурсу

while (InterlockedExchange(&resourceInUse, (long)TRUE)) Sleep(0);

//...отримали доступ:

//Interlocked-функція додавання

InterlockedExchangeAdd(&itersCur, 1);

//виводимо поточний прогрес

progress((int)ceil(100.0\*itersCur/itersNum)); Sleep(5);

//доступ вже не потрібен

InterlockedExchange(&resourceInUse, (long)FALSE);

}

// Міняємо поточний елемент з мінімальним в критичній секції

int temp = arr[i];

//входимо в критичну секцію

EnterCriticalSection(&cs);

arr[i] = arr[minValueIndex];

arr[minValueIndex] = temp;

//покидаємо критичну секцію

LeaveCriticalSection(&cs);

}

//Зберігаємо індекс потоку, який завершився

//в глобальний масив в критичній секції

//входимо в критичну секцію

EnterCriticalSection(&cs2);

threadFinals[threadFinalsIndex] = threadNum;

//Interlocked-інкремент

InterlockedIncrement(&threadFinalsIndex);

//покидаємо критичну секцію

LeaveCriticalSection(&cs2);

}

//==============================================================

//функція ініціалізації вхідного масиву випадковими числами в потоці

DWORD WINAPI InitArray(LPVOID Param) {

//заповнимо вхідний масив випадковими числами 0 <= a[i] < N\*10

srand((unsigned)time(NULL));

for(int i = 0; i < N; i++) {

a[i] = rand()%(N\*10);

//відмічаємо, що одини елемент "готовий"

ReleaseSemaphore(hSemaphore,1,NULL);

Sleep(250);

}

return 0;

}

//==============================================================

//функція копіювання посортованого масиву b у вхідний масив а

DWORD WINAPI FinalArray(LPVOID Param) {

int p = 0;

for(int i = 0; i < K; i++)

for(int j = 0; j < size[i]; j++, p++) {

//чекаємо на отримання мютекса

WaitForSingleObject(hMutex, INFINITE);

\*(a+p) = b[i][j];

Sleep(250);

//відпускаємо мютекс

ReleaseMutex(hMutex);

}

return 0;

}

//==============================================================

//функція опрацювання 1-го потоку

DWORD WINAPI Output0(LPVOID Param) {

sort(b[0], size[0], 0);

return 0;

}

//==============================================================

//функція опрацювання 2-го потоку

DWORD WINAPI Output1(LPVOID Param) {

sort(b[1], size[1], 1);

return 0;

}

//==============================================================

//функція опрацювання 3-го потоку

DWORD WINAPI Output2(LPVOID Param) {

sort(b[2], size[2], 2);

return 0;

}

//==============================================================

//функція опрацювання 4-го потоку

DWORD WINAPI Output3(LPVOID Param) {

sort(b[3], size[3], 3);

return 0;

}

//==============================================================

//функція опрацювання 5-го потоку

DWORD WINAPI Output4(LPVOID Param) {

sort(b[4], size[4], 4);

return 0;

}

//==============================================================

//функція опрацювання 6-го потоку

DWORD WINAPI Output5(LPVOID Param) {

sort(b[5], size[5], 5);

return 0;

}

//==============================================================

//функція опрацювання 7-го потоку

DWORD WINAPI Output6(LPVOID Param) {

sort(b[6], size[6], 6);

return 0;

}

//==============================================================

//функція опрацювання 8-го потоку

DWORD WINAPI Output7(LPVOID Param) {

sort(b[7], size[7], 7);

return 0;

}

int main() {

if(toScreen)

setlocale(LC\_ALL,"Ukrainian");

else if((A = fopen("d:\\output.txt", "w")) == NULL) {

return -1;

}

//створюємо семафор

hSemaphore = CreateSemaphore(NULL, 0, 10, NULL);

//створюємо потік, який готує елементи вхідного масиву

HANDLE hInitThread;

DWORD IDInitThread;

hInitThread = CreateThread(NULL, 0, InitArray, NULL, 0, &IDInitThread);

// потік main виводить елементи масиву

// тільки після їх підготовки в InitArray()

if(toScreen)

printf("\nIнiцiалiзацiя масиву a[%d]:\n", N);

else

fprintf(A, "\nIнiцiалiзацiя масиву a[%d]:\n", N);

for(int i = 0; i < N; i++) {

//чекаємо на включення семафора

WaitForSingleObject(hSemaphore, INFINITE);

//...і виводимо елемент на екран

if(toScreen)

printf("%d\t", a[i]);

else

fprintf(A, "%d\t", a[i]);

}

//закриваємо дескриптори семафору та потоку

CloseHandle(hSemaphore);

CloseHandle(hInitThread);

if(toScreen) {

printf("\nДля продовження - натиснiть клавiшу...");

\_getch();

} else {

fprintf(A, "\nДля продовження - натиснiть клавiшу...");

system("pause");

}

//знайдем макс. та мін. елементи масиву

int max = a[0];

int min = a[0];

for(int i = 1; i < N; i++) {

if(a[i] > max) max = a[i];

if(a[i] < min) min = a[i];

}

//обнулимо масив size

for(int i = 0; i < K; i++) size[i] = 0;

//крок з яким будемо рахувати межі проміжків

int step = int((max - min + 1)/K);

//розбиваємо вхідний масив на підмасиви

//для сортування їх в паралельних потоках

for(int j = 0; j < N; j++)

for(int i = 0; i < K; i++) {

//верхня межа підмасиву

int right = (i == K-1) ? (max+1) : (min+step\*(i+1));

//якщо значення елементу входить у проміжок i-го підмасиву -

//записуємо його туди

if(a[j] < right) {

b[i][size[i]++] = a[j];

break;

}

}

//розрахуємо загальну к-сть ітерацій алгоритму сортування

itersNum = 0;

for (int s = 0; s < K; s++)

for (int i = 0; i < size[s]; i++)

for (int j = i + 1; j < size[s]; j++, itersNum++);

HANDLE hThread[K]; //масив дескрипторів потоків

DWORD ThreadId[K]; //унікальні ідентифікатори потоків

//створюємо потоки

hThread[0] = CreateThread(NULL, 0, Output0, NULL, 0, &ThreadId[0]);

if(K > 1) hThread[1] = CreateThread(NULL, 0, Output1, NULL, CREATE\_SUSPENDED, &ThreadId[1]);

if(K > 2) hThread[2] = CreateThread(NULL, 0, Output2, NULL, CREATE\_SUSPENDED, &ThreadId[2]);

if(K > 3) hThread[3] = CreateThread(NULL, 0, Output3, NULL, CREATE\_SUSPENDED, &ThreadId[3]);

if(K > 4) hThread[4] = CreateThread(NULL, 0, Output4, NULL, CREATE\_SUSPENDED, &ThreadId[4]);

if(K > 5) hThread[5] = CreateThread(NULL, 0, Output5, NULL, CREATE\_SUSPENDED, &ThreadId[5]);

if(K > 6) hThread[6] = CreateThread(NULL, 0, Output6, NULL, CREATE\_SUSPENDED, &ThreadId[6]);

if(K > 7) hThread[7] = CreateThread(NULL, 0, Output7, NULL, CREATE\_SUSPENDED, &ThreadId[7]);

//ініціалізація критичних секцій

InitializeCriticalSection(&cs);

InitializeCriticalSection(&cs2);

//запуск потоків

for(int i = 0; i < K; i++) ResumeThread(hThread[i]);

//чекаємо на завершення всіх потоків

WaitForMultipleObjects(K, hThread, TRUE,INFINITE);

progress(100);

//знищення критичних секцій

DeleteCriticalSection(&cs);

DeleteCriticalSection(&cs2);

//завершуємо всі потоки в порядку їх завершення

DWORD ExitCode[K];

for(int i = 0; i < K; i++) {

//отримуємо код завершення потоків

GetExitCodeThread(hThread[i], &ExitCode[i]);

//завершення потоку

TerminateThread(hThread[i], ExitCode[i]);

//закриття дексриптора потоку

CloseHandle(hThread[i]);

}

if(toScreen) {

//створюємо мютекс

hMutex = CreateMutex(NULL, FALSE, NULL);

//створюємо потік, який копіює посортований масив b у вхідний масив а

HANDLE hFinalThread;

DWORD IDFinalThread;

hFinalThread = CreateThread(NULL, 0, FinalArray, NULL, 0, &IDFinalThread);

//затримаємо потік main, щоб потік FinalArray() першим отримав мютекс

Sleep(200);

printf("\n\nМасив a[%d] пiсля сортування:\n", N);

for(int i = 0; i < N; i++) {

//отримаємо мютекс

WaitForSingleObject(hMutex, INFINITE);

printf("%d\t", a[i]);

//відпускаємо мютекс

ReleaseMutex(hMutex);

}

//закриваємо дескриптори мютекса та потоку

CloseHandle(hMutex);

CloseHandle(hFinalThread);

printf("\nДля завершення програми - натиснiть клавiшу...");

\_getch();

} else {

fprintf(A, "\n\nМасив a[%d] пiсля сортування:\n", N);

for(int i = 0; i < K; i++)

for(int j = 0; j < size[i]; j++)

fprintf(A, "%d\t", b[i][j]);

}

return 0;

}

//функції, що експортуються

namespace MathFuncs {

//для виклику, при ранньому зв'язуванні

int mainDLL() {

toScreen = 1;

return main();

}

//для виклику з rundll32

void CALLBACK OutputFile(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow){

toScreen = 0;

main();

}

//для виклику, при пізньому зв'язуванні

void CALLBACK OutputScreen(HWND hwnd, HINSTANCE hinst, LPSTR pszCmdLine, int nCmdShow){

toScreen = 1;

main();

}

}

## Результат виконання програми

Запустимо **batDll.bat** на виконання (рис. 4.1).

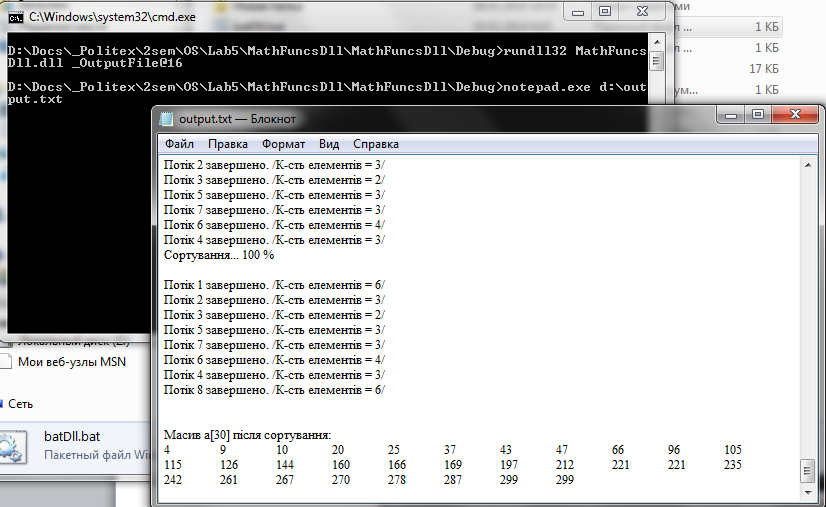


Рис. 4.1. Запуск DLL з rundll32.exe

Запустимо **MyExecRefsDll.exe** на виконання (рис. 4.2).

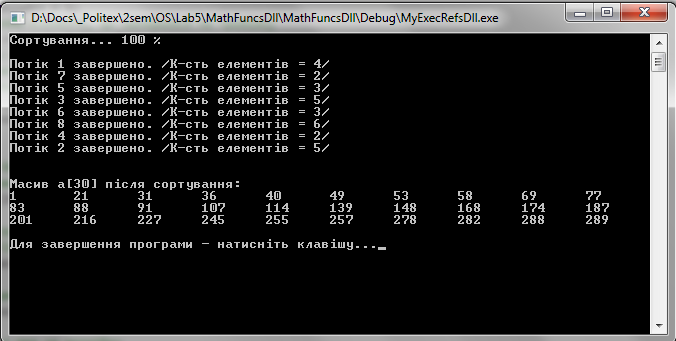


Рис. 4.2. Статичний зв’язок між DLL та програмою

Запустимо **DynamicLinking.exe** на виконання (рис. 4.3).

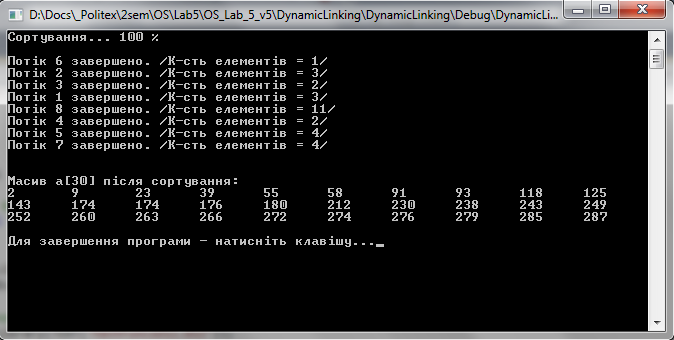


Рис. 4.3. Динамічний зв’язок між DLL та програмою

Як бачимо, у всіх випадках алгоритм програми працює правильно.

## ВИСНОВКИ

На даній лабораторній роботі я ознайомився з динамічно-зв’язувальними бібліотеками (Dynamic-link library) в ОС Windows. Навчитися реалізовувати динамічно-зв’язувальні бібліотеки. Запускати їх функціонал з інших програм.