

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Практична робота №1**

**з дисципліни «Сучасні освітні технології »   
на тему:**

**«**Сучасне системне програмне забезпечення»

Виконав:

студент гр. ПЗ2421

Кулик С. В.

Прийняв:

Жеваго О. О.

Дніпро, 2025

Тема. Дослідження часових характеристик різних способів копіювання файлів у Windows.

Мета. Ознайомитися зі способами копіювання файлів у Windows. Дослідити часові характеристики різних способів копіювання файлів. Визначити найбільш ефективний спосіб для файлів різного розміру.

**Постановка задачі**

Необхідно створити програму, яка реалізує три різні способи копіювання файлів:

1. За допомогою функції CopyFile (Win32 API)
2. За допомогою бібліотеки C (fread, fwrite)
3. За допомогою Win API (ReadFile, WriteFile)

Програма повинна виміряти час копіювання файлів різного розміру (1 МБ, 100 МБ, 1 ГБ) та визначити найефективніший метод.

**Лістинг програми**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <iostream>

#include <fstream>

#include <chrono>

#include <string>

#include <iomanip>

#include <vector>

using namespace std;

using namespace std::chrono;

template <typename T>

T myMin(std::initializer\_list<T> values) {

auto it = values.begin();

T result = \*it;

++it;

for (; it != values.end(); ++it) {

if (\*it < result) result = \*it;

}

return result;

}

// Функція для копіювання файлу за допомогою CopyFile

bool CopyFileMethod(const string& source, const string& destination, double& timeMs) {

auto start = high\_resolution\_clock::now();

BOOL result = CopyFileA(source.c\_str(), destination.c\_str(), FALSE);

auto end = high\_resolution\_clock::now();

timeMs = duration\_cast<microseconds>(end - start).count() / 1000.0;

return result != 0;

}

// Функція для копіювання файлу за допомогою бібліотеки C (fread, fwrite)

bool CopyFileC(const string& source, const string& destination, double& timeMs) {

auto start = high\_resolution\_clock::now();

FILE\* srcFile = fopen(source.c\_str(), "rb");

if (!srcFile) {

timeMs = 0;

return false;

}

FILE\* destFile = fopen(destination.c\_str(), "wb");

if (!destFile) {

fclose(srcFile);

timeMs = 0;

return false;

}

const size\_t BUFFER\_SIZE = 64 \* 1024; // 64KB buffer

char\* buffer = new char[BUFFER\_SIZE];

size\_t bytesRead;

bool success = true;

while ((bytesRead = fread(buffer, 1, BUFFER\_SIZE, srcFile)) > 0) {

if (fwrite(buffer, 1, bytesRead, destFile) != bytesRead) {

success = false;

break;

}

}

delete[] buffer;

fclose(srcFile);

fclose(destFile);

auto end = high\_resolution\_clock::now();

timeMs = duration\_cast<microseconds>(end - start).count() / 1000.0;

return success;

}

// Функція для копіювання файлу за допомогою Win API (ReadFile, WriteFile)

bool CopyFileWinAPI(const string& source, const string& destination, double& timeMs) {

auto start = high\_resolution\_clock::now();

HANDLE srcFile = CreateFileA(source.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ,

NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (srcFile == INVALID\_HANDLE\_VALUE) {

timeMs = 0;

return false;

}

HANDLE destFile = CreateFileA(destination.c\_str(), GENERIC\_WRITE, 0,

NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (destFile == INVALID\_HANDLE\_VALUE) {

CloseHandle(srcFile);

timeMs = 0;

return false;

}

const DWORD BUFFER\_SIZE = 64 \* 1024; // 64KB buffer

char\* buffer = new char[BUFFER\_SIZE];

DWORD bytesRead, bytesWritten;

bool success = true;

while (ReadFile(srcFile, buffer, BUFFER\_SIZE, &bytesRead, NULL) && bytesRead > 0) {

if (!WriteFile(destFile, buffer, bytesRead, &bytesWritten, NULL) ||

bytesWritten != bytesRead) {

success = false;

break;

}

}

delete[] buffer;

CloseHandle(srcFile);

CloseHandle(destFile);

auto end = high\_resolution\_clock::now();

timeMs = duration\_cast<microseconds>(end - start).count() / 1000.0;

return success;

}

// Функція для створення тестового файлу заданого розміру

bool CreateTestFile(const string& filename, size\_t sizeMB) {

ofstream file(filename, ios::binary);

if (!file) return false;

const size\_t CHUNK\_SIZE = 1024 \* 1024; // 1MB chunks

char\* buffer = new char[CHUNK\_SIZE];

// Заповнюємо буфер псевдовипадковими даними

for (size\_t i = 0; i < CHUNK\_SIZE; i++) {

buffer[i] = static\_cast<char>(i % 256);

}

for (size\_t i = 0; i < sizeMB; i++) {

file.write(buffer, CHUNK\_SIZE);

if (!file.good()) {

delete[] buffer;

file.close();

return false;

}

}

delete[] buffer;

file.close();

return true;

}

// Функція для отримання розміру файлу

long long GetFileSize(const string& filename) {

HANDLE file = CreateFileA(filename.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ,

NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (file == INVALID\_HANDLE\_VALUE) return -1;

LARGE\_INTEGER fileSize;

if (!GetFileSizeEx(file, &fileSize)) {

CloseHandle(file);

return -1;

}

CloseHandle(file);

return fileSize.QuadPart;

}

// Функція для видалення файлу

void DeleteTestFile(const string& filename) {

DeleteFileA(filename.c\_str());

}

int main() {

// Налаштування консолі для українського тексту

SetConsoleOutputCP(1251); // вывод UTF-8

SetConsoleCP(1251); // ввод UTF-8

cout << "=== ЛАБОРАТОРНА РОБОТА №1 ===" << endl;

cout << "Дослідження часових характеристик різних способів копіювання файлів у Windows" << endl;

cout << "=====================================================" << endl << endl;

// Розміри файлів для тестування (в МБ)

vector<size\_t> fileSizes = { 1, 100, 1000 }; // 1MB, 100MB, 1GB

cout << left << setw(12) << "Розмір файлу"

<< setw(18) << "CopyFile (мс)"

<< setw(18) << "C Library (мс)"

<< setw(18) << "Win API (мс)" << endl;

cout << string(70, '-') << endl;

for (size\_t size : fileSizes) {

string sourceFile = "test\_" + to\_string(size) + "MB.dat";

string destFile1 = "copy1\_" + to\_string(size) + "MB.dat";

string destFile2 = "copy2\_" + to\_string(size) + "MB.dat";

string destFile3 = "copy3\_" + to\_string(size) + "MB.dat";

cout << "Створення тестового файлу " << size << " МБ... ";

cout.flush();

if (!CreateTestFile(sourceFile, size)) {

cout << "ПОМИЛКА!" << endl;

continue;

}

cout << "OK" << endl;

// Перевіряємо розмір створеного файлу

long long actualSize = GetFileSize(sourceFile);

cout << "Фактичний розмір файлу: " << actualSize / (1024.0 \* 1024.0) << " МБ" << endl;

double time1, time2, time3;

bool success1, success2, success3;

// Тест 1: CopyFile

cout << "Тестування CopyFile... ";

cout.flush();

success1 = CopyFileMethod(sourceFile, destFile1, time1);

cout << (success1 ? "OK" : "ПОМИЛКА") << endl;

// Тест 2: C Library

cout << "Тестування C Library... ";

cout.flush();

success2 = CopyFileC(sourceFile, destFile2, time2);

cout << (success2 ? "OK" : "ПОМИЛКА") << endl;

// Тест 3: Win API

cout << "Тестування Win API... ";

cout.flush();

success3 = CopyFileWinAPI(sourceFile, destFile3, time3);

cout << (success3 ? "OK" : "ПОМИЛКА") << endl;

// Виводимо результати

cout << left << setw(12) << (to\_string(size) + " МБ")

<< setw(18) << (success1 ? to\_string(time1) : "ПОМИЛКА")

<< setw(18) << (success2 ? to\_string(time2) : "ПОМИЛКА")

<< setw(18) << (success3 ? to\_string(time3) : "ПОМИЛКА") << endl;

// Аналіз швидкості

if (success1 && success2 && success3) {

double minTime = myMin({ time1, time2, time3 });

cout << "Найшвидший метод: ";

if (time1 == minTime) cout << "CopyFile";

else if (time2 == minTime) cout << "C Library";

else cout << "Win API";

cout << " (" << minTime << " мс)" << endl;

// Швидкість копіювання в МБ/с

double speed1 = size / (time1 / 1000.0);

double speed2 = size / (time2 / 1000.0);

double speed3 = size / (time3 / 1000.0);

cout << "Швидкості копіювання:" << endl;

cout << " CopyFile: " << fixed << setprecision(2) << speed1 << " МБ/с" << endl;

cout << " C Library: " << speed2 << " МБ/с" << endl;

cout << " Win API: " << speed3 << " МБ/с" << endl;

}

// Очищення файлів

DeleteTestFile(sourceFile);

DeleteTestFile(destFile1);

DeleteTestFile(destFile2);

DeleteTestFile(destFile3);

cout << endl;

}

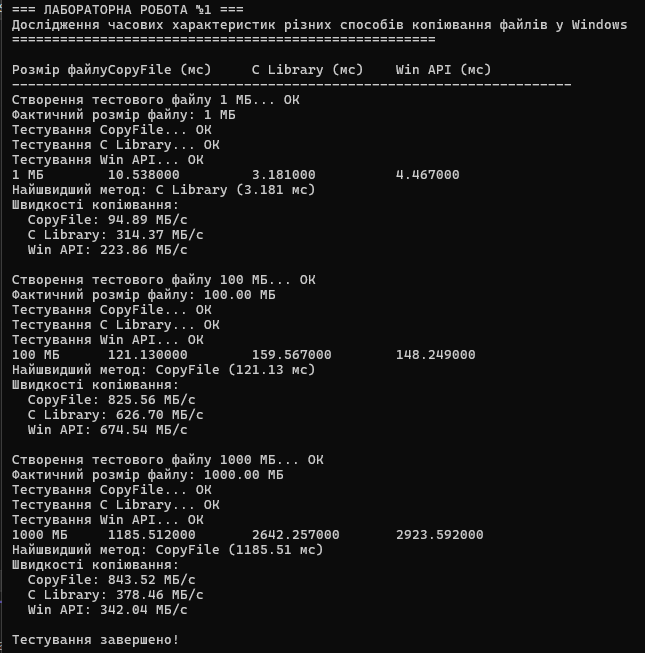
cout << "Тестування завершено!" << endl;

cout << "\nНатисніть Enter для виходу...";

cin.get();

return 0;

}

**Часові характеристики копіювання файлів  
**

**Аналіз результатів та висновки**

**Аналіз за розмірами файлів:**

**1. Малі файли (1 МБ):**

* **C Library** показала найкращий результат (3.18 мс, 314.37 МБ/с)
* **Win API** посідає друге місце (4.47 мс, 223.86 МБ/с)
* **CopyFile** виявилась найповільнішою (10.54 мс, 94.89 МБ/с)

**2. Середні файли (100 МБ):**

* **CopyFile** стає лідером (121.13 мс, 825.56 МБ/с)
* **Win API** на другому місці (148.25 мс, 674.54 МБ/с)
* **C Library** найповільніша (159.57 мс, 626.70 МБ/с)

**3. Великі файли (1 ГБ):**

* **CopyFile** залишається найефективнішим (1185.51 мс, 843.52 МБ/с)
* **C Library** на другому місці (2642.26 мс, 378.46 МБ/с)
* **Win API** найповільніший (2923.59 мс, 342.04 МБ/с)

**Висновки про ефективність:**

1. **CopyFile** демонструє найкращу масштабованість - чим більший файл, тим більша її перевага. Для великих файлів вона в 2.2-2.5 рази швидша за конкурентів завдяки оптимізації на рівні ОС Windows.
2. **C Library** ефективна для малих файлів через мінімальні накладні витрати, але погано масштабується - її продуктивність падає з ростом розміру файлів через додаткові рівні буферизації.
3. **Win API** показує стабільну середню продуктивність на всіх розмірах, але програє CopyFile через відсутність системних оптимізацій, притаманних високорівневій функції CopyFile.
4. **Парадокс малих файлів**: CopyFile програє на 1 МБ через значні накладні витрати на ініціалізацію системних механізмів оптимізації, які виправдовують себе тільки на більших обсягах даних.