МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Практична робота №3**

**з дисципліни «Сучасне системне програмне забезпечення»   
на тему:**

**«**Асинхронні операції введення/виведення в OC Windows.»

Виконав:

студент гр. ПЗ2421

Кулик С. В.

Прийняв:

Жеваго О. О.

Дніпро, 2025

Тема. Асинхронні операції введення/виведення в OC Windows.

Мета. Здобути практичні навички організації операцій введення-виводу з перекриттям.

**Постановка задачі**

Скласти програму для копіювання файлу, використовуючи введення/виведення з перекриттям. Продемонструвати відмінність від синхронних операцій введення/виведення;

**Текст программи**

#include <iostream>

#include <windows.h>

#include <chrono>

#include <stdexcept>

#include <string>

// --- ШЛЯХИ ФАЙЛІВ ---

const std::wstring SOURCE\_PATH = L"D:\\test.bin";

const std::wstring DEST\_ASYNC\_PATH = L"D:\\copy\_async.bin";

const std::wstring DEST\_SYNC\_PATH = L"D:\\copy\_sync.bin";

// --- --- --- --- --- --- --- --- --- --- ---

// Розмір буфера для копіювання (1 МБ)

#define BUFFER\_SIZE (1024 \* 1024)

// Функція для обробки помилок Windows API

void HandleError(const wchar\_t\* operation) {

DWORD error = GetLastError();

if (error == ERROR\_IO\_PENDING) return;

wchar\_t\* messageBuffer = nullptr;

FormatMessageW(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM | FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL, error, MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), (LPWSTR)&messageBuffer, 0, NULL

);

std::wcerr << L"Помилка під час " << operation << L" (" << error << L"): "

<< (messageBuffer ? messageBuffer : L"Невідома помилка") << L"\n";

if (messageBuffer) {

LocalFree(messageBuffer);

}

}

// --- 1. СИНХРОННА ФУНКЦІЯ КОПІЮВАННЯ ---

void CopyFileSync(const std::wstring& sourcePath, const std::wstring& destPath) {

std::wcout << L"СИНХРОННИЙ режим: Початок копіювання " << sourcePath << L" -> " << destPath << std::endl;

HANDLE hSource = INVALID\_HANDLE\_VALUE;

HANDLE hDest = INVALID\_HANDLE\_VALUE;

BYTE\* buffer = nullptr;

try {

// Відкриття файлів без прапора FILE\_FLAG\_OVERLAPPED

hSource = CreateFileW(sourcePath.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hSource == INVALID\_HANDLE\_VALUE) throw std::runtime\_error("Не вдалося відкрити джерело.");

hDest = CreateFileW(destPath.c\_str(), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hDest == INVALID\_HANDLE\_VALUE) throw std::runtime\_error("Не вдалося створити призначення.");

buffer = new BYTE[BUFFER\_SIZE];

DWORD bytesRead = 0;

DWORD bytesWritten = 0;

LONGLONG currentOffset = 0;

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

// КОМЕНТАР: Потік БЛОКУЄТЬСЯ, чекаючи завершення ReadFile

if (!WriteFile(hDest, buffer, bytesRead, &bytesWritten, NULL)) {

throw std::runtime\_error("Помилка запису файлу.");

}

// КОМЕНТАР: Потік БЛОКУЄТЬСЯ, чекаючи завершення WriteFile

currentOffset += bytesWritten;

}

std::wcout << L"СИНХРОННЕ копіювання завершено. Всього скопійовано: "

<< currentOffset / (1024.0 \* 1024.0) << L" МБ." << std::endl;

}

catch (const std::exception& e) {

std::cerr << "Синхронна помилка: " << e.what() << std::endl;

HandleError(L"синхронної операції");

}

if (buffer) delete[] buffer;

if (hSource != INVALID\_HANDLE\_VALUE) CloseHandle(hSource);

if (hDest != INVALID\_HANDLE\_VALUE) CloseHandle(hDest);

}

// --- 2. АСИНХРОННА ФУНКЦІЯ КОПІЮВАННЯ (З ПЕРЕКРИТТЯМ) ---

void CopyFileAsync(const std::wstring& sourcePath, const std::wstring& destPath) {

std::wcout << L"АСИНХРОННИЙ режим: Початок копіювання " << sourcePath << L" -> " << destPath << std::endl;

HANDLE hSource = INVALID\_HANDLE\_VALUE;

HANDLE hDest = INVALID\_HANDLE\_VALUE;

BYTE\* buffer = nullptr;

OVERLAPPED overlapped = { 0 };

try {

// Відкриття файлів з обов'язковим прапором FILE\_FLAG\_OVERLAPPED

hSource = CreateFileW(sourcePath.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED, NULL);

if (hSource == INVALID\_HANDLE\_VALUE) throw std::runtime\_error("Не вдалося відкрити джерело.");

hDest = CreateFileW(destPath.c\_str(), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED, NULL);

if (hDest == INVALID\_HANDLE\_VALUE) throw std::runtime\_error("Не вдалося створити призначення.");

buffer = new BYTE[BUFFER\_SIZE];

DWORD bytesRead;

DWORD bytesWritten;

LONGLONG currentOffset = 0;

// Створення Event для OVERLAPPED структури

overlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

if (overlapped.hEvent == NULL) throw std::runtime\_error("Не вдалося створити Event.");

// Основний цикл копіювання

while (true) {

overlapped.Offset = (DWORD)(currentOffset & 0xFFFFFFFF);

overlapped.OffsetHigh = (DWORD)(currentOffset >> 32);

ResetEvent(overlapped.hEvent);

// Асинхронне ЧИТАННЯ

BOOL success = ReadFile(hSource, buffer, BUFFER\_SIZE, NULL, &overlapped);

if (!success && GetLastError() == ERROR\_IO\_PENDING) {

// КОМЕНТАР: Операція виконується ЯДРОМ ОС. Потік ЦП чекає тільки на Event

if (!GetOverlappedResult(hSource, &overlapped, &bytesRead, TRUE)) {

if (GetLastError() == ERROR\_HANDLE\_EOF) break;

throw std::runtime\_error("Помилка очікування читання.");

}

}

else if (!success) {

HandleError(L"читання файлу");

break;

}

else {

bytesRead = BUFFER\_SIZE;

}

if (bytesRead == 0) break; // Кінець файлу

// Асинхронний ЗАПИС

ResetEvent(overlapped.hEvent);

success = WriteFile(hDest, buffer, bytesRead, NULL, &overlapped);

if (!success && GetLastError() == ERROR\_IO\_PENDING) {

// КОМЕНТАР: Операція виконується ЯДРОМ ОС. Потік ЦП чекає тільки на Event

if (!GetOverlappedResult(hDest, &overlapped, &bytesWritten, TRUE)) {

throw std::runtime\_error("Помилка очікування запису.");

}

}

else if (!success) {

HandleError(L"запис файлу");

break;

}

else {

bytesWritten = bytesRead;

}

currentOffset += bytesWritten;

std::wcout << L"\rСкопійовано: " << currentOffset / (1024.0 \* 1024.0) << L" МБ..." << std::flush;

}

std::wcout << L"\rАСИНХРОННЕ копіювання завершено. Всього скопійовано: "

<< currentOffset / (1024.0 \* 1024.0) << L" МБ." << std::endl;

}

catch (const std::exception& e) {

std::cerr << "Асинхронна помилка: " << e.what() << std::endl;

HandleError(L"асинхронної операції");

}

if (overlapped.hEvent != NULL) CloseHandle(overlapped.hEvent);

if (buffer) delete[] buffer;

if (hSource != INVALID\_HANDLE\_VALUE) CloseHandle(hSource);

if (hDest != INVALID\_HANDLE\_VALUE) CloseHandle(hDest);

}

int wmain() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::wcout.imbue(std::locale(""));

std::wcout << L"=======================================================" << std::endl;

std::wcout << L"ТЕСТ КОПІЮВАННЯ ФАЙЛУ (СИНХРОННО vs АСИНХРОННО)" << std::endl;

std::wcout << L"Джерело: " << SOURCE\_PATH << std::endl;

std::wcout << L"=======================================================" << std::endl;

// 1. СИНХРОННЕ копіювання

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

CopyFileSync(SOURCE\_PATH, DEST\_SYNC\_PATH);

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

auto duration\_sync = std::chrono::duration\_cast<std::chrono::milliseconds>(end\_sync - start\_sync);

std::wcout << L"\nЧас СИНХРОННОГО копіювання: " << duration\_sync.count() << L" мс." << std::endl;

std::wcout << L"-------------------------------------------------------" << std::endl;

// 2. АСИНХРОННЕ копіювання

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

CopyFileAsync(SOURCE\_PATH, DEST\_ASYNC\_PATH);

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

auto duration\_async = std::chrono::duration\_cast<std::chrono::milliseconds>(end\_async - start\_async);

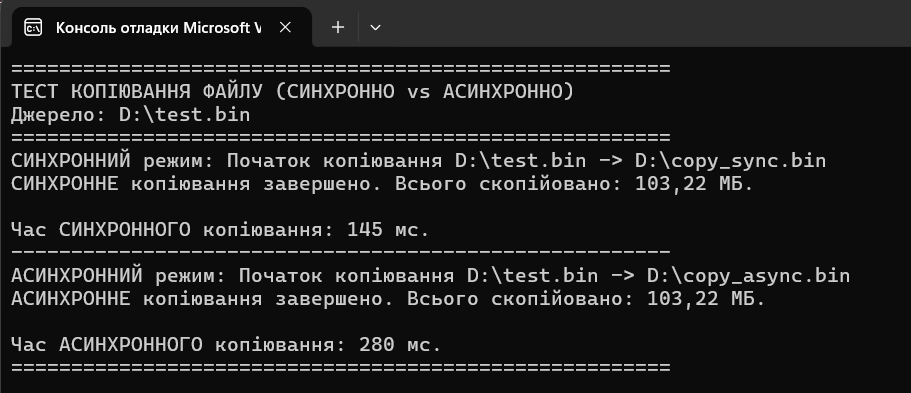
std::wcout << L"\nЧас АСИНХРОННОГО копіювання: " << duration\_async.count() << L" мс." << std::endl;

std::wcout << L"=======================================================" << std::endl;

return 0;

}

**Результати роботи**

****

**Висновки**

У ході лабораторної роботи було реалізовано копіювання файлу, використовуючи механізми **синхронного** та **асинхронного (з перекриттям)** введення/виведення (I/O) на платформі Windows, що дозволило виявити наступні фундаментальні особливості:

|  |  |  |
| --- | --- | --- |
| Характеристика | Синхронне I/O | Асинхронне I/O (з перекриттям) |
| Основна дія | Виклик функції блокує потік ЦП, доки операція I/O не завершиться. | Виклик функції не блокує потік, негайно повертаючи керування (з кодом ERROR\_IO\_PENDING). |
| Робота ЦП | Потік ЦП знаходиться у стані очікування протягом усього часу, поки апаратне забезпечення (диск, мережа) виконує передачу даних. | Потік ЦП може продовжувати роботу (обчислення або ініціювати інший I/O), поки апаратна операція виконується автономно через DMA та I/O Manager. |
| Ключовий елемент | Жодних додаткових структур. | Обов'язкова структура OVERLAPPED для передачі контексту (зміщення, Event). |

**Умови, за яких асинхронність повільніша за синхронність (Спостереження за результатом)**

Незважаючи на теоретичну перевагу, на практиці, особливо у випадках простого копіювання, **асинхронне копіювання часто виявляється повільнішим** з наступних причин:

1. **Надмірні накладні витрати на рівні ядра (I/O Overhead):** Для кожного асинхронного запиту ядро ОС повинно виконати значно більше роботи: ініціалізувати структуру OVERLAPPED, створити об'єкт події (Event) та сформувати пакет запиту I/O (**IRP**). На швидких дисках (SSD) або при використанні кешу ці додаткові витрати на підготовку I/O-запиту перевищують час, зекономлений на асинхронності.
2. **Відсутність справжнього перекриття в коді:** У даній реалізації (де ми викликаємо GetOverlappedResult(..., TRUE) одразу після ReadFile та WriteFile) потік, хоч і використовує асинхронний механізм, все одно **чекає** завершення кожної операції послідовно. Це створює шаблон "синхронний-на-асинхронному", де накладні витрати є, а вигоди паралелізму — немає. Для виграшу в часі потрібна **подвійна буферизація** (ініціація читання наступного блоку паралельно із записом поточного).
3. **Взаємодія з кешем файлової системи:** Синхронні операції ефективно працюють із вбудованим кешем ОС. Асинхронні операції, особливо ті, що використовують прапори, призначені для високої продуктивності (наприклад, FILE\_FLAG\_NO\_BUFFERING — не використовувався у нашій програмі, але часто застосовується в асинхронних програмах), можуть обходити кеш, що призводить до повільнішого доступу безпосередньо до фізичного пристрою.

**Умови, за яких асинхронність швидша або необхідна**

Незважаючи на повільнішу роботу в простому тесті, асинхронне I/O є **незамінним** для:

1. **Високонавантажених серверів:** Дозволяє одному потоку одночасно обробляти сотні запитів, не блокуючись на повільних операціях I/O (особливо з використанням **портів завершення I/O - I/O Completion Ports**).
2. **Повільні пристрої:** На старих HDD, мережевих з'єднаннях або при роботі з віддаленими файлами (де час очікування великий), асинхронність дозволяє ЦП виконувати іншу корисну роботу, поки чекає на завершення I/O.
3. **Підвищення відгуку інтерфейсу:** У програмах з графічним інтерфейсом (GUI) асинхронний I/O запобігає "зависанню" інтерфейсу, оскільки тривалі операції не блокують основний потік повідомлень.

Асинхронне введення/виведення — це не завжди про **абсолютну швидкість**, а в першу чергу про **ефективність використання ресурсів** та **уникнення блокування потоку** для досягнення кращої масштабованості та відгуку системи.