Лабораторна робота №1

Тема: Процеси та потоки

Мета: Взаємодія між процесами. Розподіл даних між процесами. Робота з файлами які відображуються у пам'ять.

Завдання 1

Необхідно написати дві програми (**три**), які будуть мати спільні дані та одночасно до них звертатися.

Існує кілька механізмів реалізації спільного доступу до даних різних процесів. Скористаємося одним з них, найбільш зручним - проектуванням файлу в пам'ять.

Одна програма буде сортувати дані у файлі, а інша відображати вміст цього файлу. Працювати обидва процеси будуть одночасно. **Третя програма** буде створювати (або заповнювати по новому) масив випадкових чисел.

Створіть файл data.dat. У ньому мають бути записані числа, згенеровані випадковим чином. Кількість чисел - 20-30 штук. Діапазон значень: від 10 до 100. (Це саме числа, а не символьні рядки зберігають ASCII коди цифр !!!)

Програма №1. "Сортування даних" (консольна)

Беремо за основу програму "Hello windows". Включаємо обробку події натискання клавіші, і відстежуємо в ньому натискання пробілу. Якщо користувач натиснув пробіл, значить починаємо сортування даних.

Виконуємо проектування файлу в пам'ять. Використовуємо для цього створений файл data.dat. В результаті отримаємо доступ до даних як до звичайного одновимірного масиву. Виконуємо сортування масиву, будь-яким з методів сортування. Вставте 1-но секундну затримку для кожної ітерації сортування масиву, це дозволить потім наочніше побачити процес сортування.

По закінченню сортування, програма виводить у вікно, рядок «Робота завершена».

Програма №2. «Виведення файлу даних у вікно» (віконна)

Виконуємо проектування файлу в пам'ять. Використовуємо для цього створений файл data.dat. В результаті отримаємо доступ до даних як до звичайного одновимірного масиву. Цей же файл проектує в пам'ять попередня програма. Створюємо таймер на 0.5 секунди. При отриманні повідомлення від таймера, виконуємо висновок всього масиву в вікно. Передбачте коректний перевивід даних у вікно, без накладень. У вікно виводиться не числа з масиву, а рядки одного і того ж символу, наприклад «*», в кількості, що дорівнює числу з масиву.

Запускаємо на виконання обидві програми одночасно. Коли друга програма запустилася і виконує висновок даних у вікно (виводить поки одну й ту ж саму картинку кожні пів секунди), натискаємо пробіл в першій програмі і вона починає сортувати масив. При цьому, так як вони дані беруть з одного і того ж файлу (обидві проектували його собі на згадку), то перша вносить зміни переставляючи дані при сортуванні, а друга виводить з себе у вікно і ми бачимо хід процесу сортування. Тимчасову затримку в першій програмі можна при потребі збільшити.

Ці дві програми демонструють можливість організації спільного доступу процесів до одних і тих самих даних. Так само демонструється механізм проектування файлу в пам'ять, як один з найкращих методів доступу до файлу.

Завдання 2.

Для коректної роботи зі спільними даними у цих двох програмах потрібно додати **синхронізацію потоків**, які можуть одночасно звертатися до спільних даних.

Для організації такої синхронізації потрібно використати об'єкт ядра ОС **mutex або semaphor**, або інший синхронізуючий об'єкт, а також **функції очікування** (наприклад, **WaitForSinglJbject()**).

Також обов'язковим є використання **обробки виняткових ситуацій** в роботі вище описаних трьох програм. Бо, некоректна робота будь якої з трьох, викличе неправильну роботу інших, через блокування спільних даних.

Для обробки виняткових ситуацій, необхідно правильно визначити **критичні секції коду** усіх написаних програм.

Результат виконання:

Програма №1 — консольний застосунок, який сортує масив цілих чисел.

Програма №2 — віконний застосунок, який зчитує числа та відображає їх у вигляді рядків з символів '*'.

Програма №3 — генератор, який створює файл із випадковими цілими числами у двійковому форматі.

Типи та призначення глобальних змінних:

data int* - вказівник на область пам’яті, де відображено вміст файлу data.dat.

hwndGlobal HWND - ідентифікатор вікна. Збережений для можливих дій над вікном.

FILE_NAME #define (char[]) - ім’я файлу, з якого читаються числові дані.

ARRAY_SIZE #define (int) - кількість чисел у масиві (розмір даних).

TIMER_ID #define (int) - ідентифікатор таймера, який оновлює вікно.

TIMER_INTERVAL #define (int) - інтервал оновлення таймера (в мілісекундах).

Методи та прийоми, використані при вирішенні задачі:

- Використання пам’яті, відображеної на файл (Memory-Mapped File);
- Синхронізація доступу до спільних ресурсів за допомогою Mutex;
- Графічний інтерфейс користувача (GUI) з використанням WinAPI;
- Оновлення інтерфейсу за таймером (SetTimer, WM_TIMER);
- Обробка помилок та перевірки доступу (if, try-catch).

Структура програм:

generator.cpp – Генератор випадкових чисел

Основні компоненти:

- generateRandomData(...) – функція генерації випадкових чисел.
- CreateMutexA(...) – створення або відкриття глобального м’ютексу "Global\\MyDataMutex" для синхронізації.
- std::ofstream – запис чисел у файл.
- std::mt19937 + std::uniform_int_distribution – генерація випадкових чисел.

lab1.cpp – Сортування з візуальною затримкою

Основні компоненти:

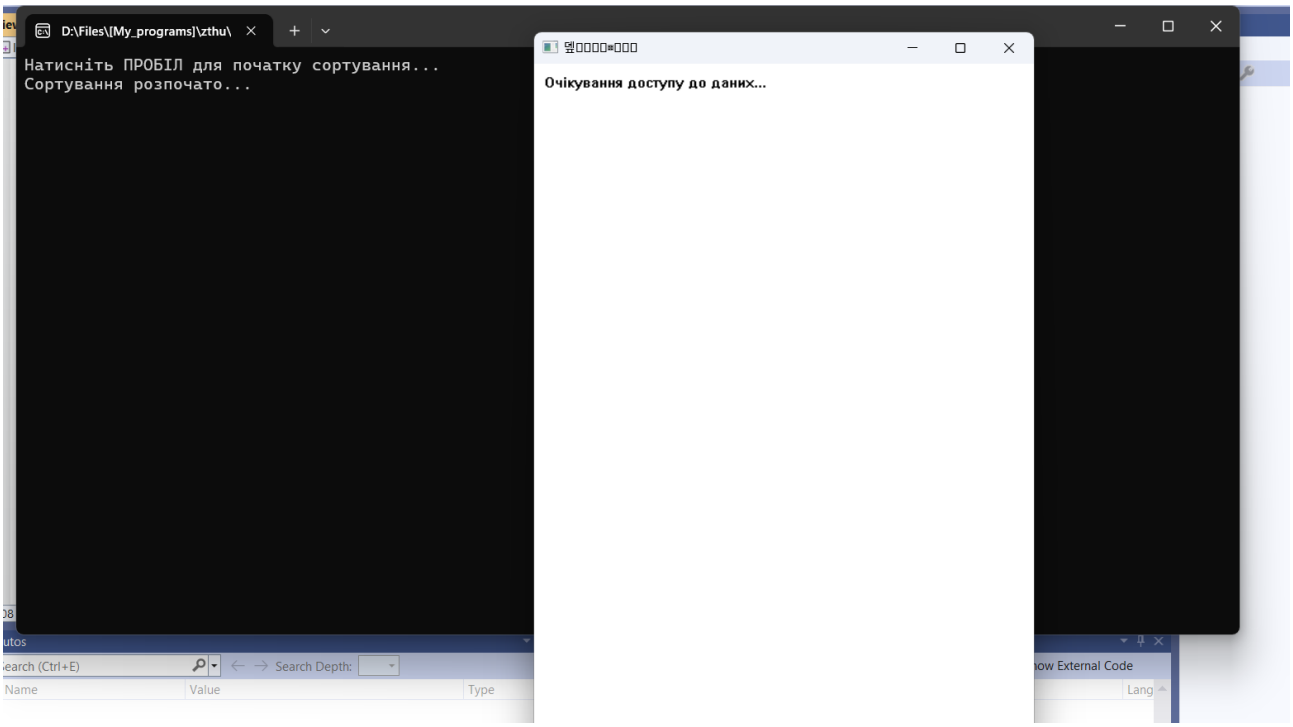
- bubbleSort(...) – сортування масиву в пам’яті.
- MapViewOfFile(...) – робота з файлом через відображення у пам’ять (data.dat).
- HANDLE + CreateFileA(...) – відкриття файлу.
- waitForSpace(...) – очікування натискання клавіші для старту.
- Mutex – забезпечення виключного доступу до даних.
- std::this_thread::sleep_for(...) – затримка для кожної ітерації сортування (1 секунда).

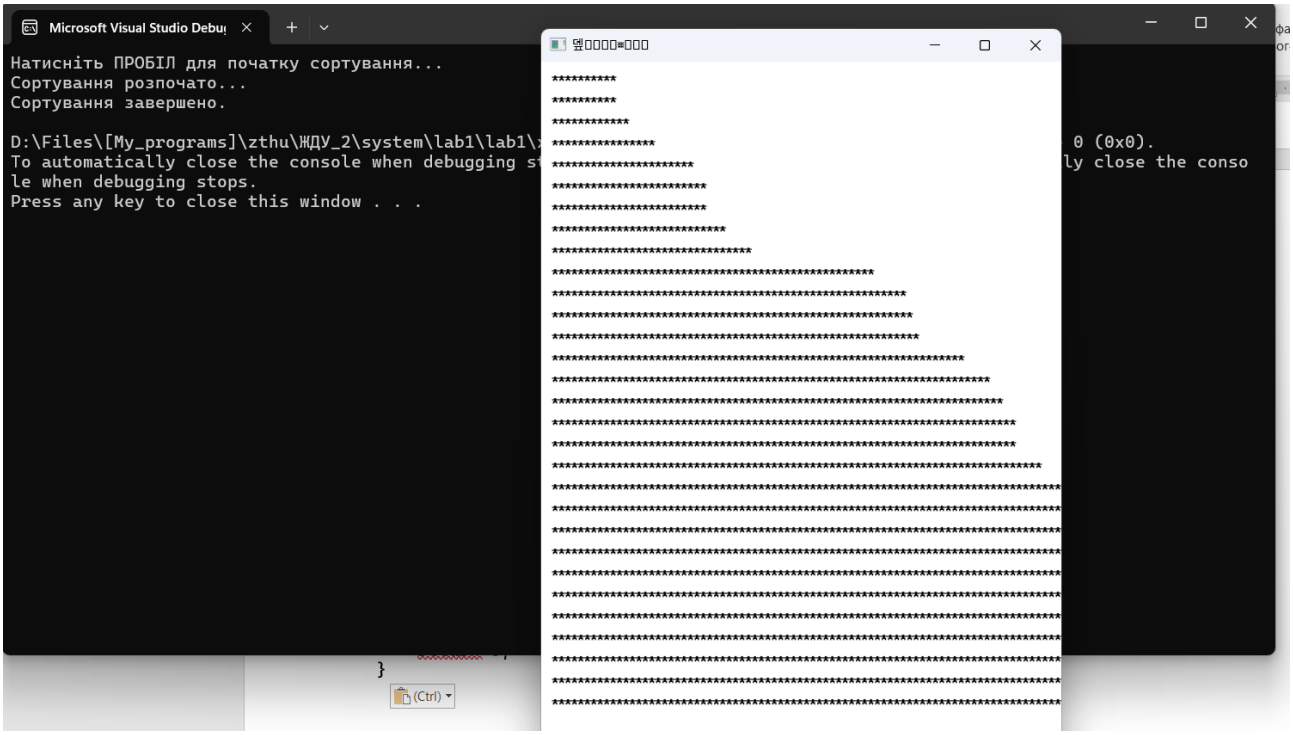
view.cpp – Графічна візуалізація даних

Основні компоненти:

- mapDataFile() – відображення файлу для читання.
- drawArrayBars(...) – виведення зірочок відповідно до значень у масиві.
- OpenMutexA(...) – синхронізований доступ до масиву через м’ютекс.
- SetTimer() + WM_TIMER – періодичне оновлення інтерфейсу (кожні 500 мс).
- TextOutA(...) – вивід рядків на екран.
- WndProc(...) – обробка повідомлень Windows.

Результат:





Лістінг generator.cpp:

```
#include <windows.h>
#include <iostream>
#include <fstream>
#include <random>
#include <ctime>
#define FILE_NAME "data.dat"
#define ARRAY_SIZE 30
// Генерація
void generateRandomData(const std::string& fileName, int count) {
    std::ofstream file(fileName, std::ios::binary);
    if (!file) {
        throw std::runtime_error("Не вдалося створити файл.");
    }
```

```
    // Ініціалізація генератора
    std::mt19937 rng(static_cast<unsigned>(time(nullptr)));
    std::uniform_int_distribution<int> dist(10, 100); // числа від 10 до 100

    // Запис у файл
    for (int i = 0; i < count; ++i) {
        int number = dist(rng);
        file.write(reinterpret_cast<const char*>(&number), sizeof(int));
    }

    file.close();
    std::cout << "Файл згенеровано з " << count << " числами!";
}

int main() {
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);

    // м'ютекс
    HANDLE hMutex = CreateMutexA(NULL, FALSE, "Global\\MyDataMutex");
    if (!hMutex) {
        std::cerr << "Не вдалося створити mutex!" << std::endl;
        return 1;
    }
    if (WaitForSingleObject(hMutex, INFINITE) == WAIT_OBJECT_0) {
        try {
            generateRandomData(FILE_NAME, ARRAY_SIZE);
        }
        catch (const std::exception& e) {
            std::cerr << "Помилка при записі у файл: " << e.what() << std::endl;
        }
        ReleaseMutex(hMutex);
    }
    else {
        std::cerr << "Не вдалося отримати mutex для запису!" << std::endl;
    }
    CloseHandle(hMutex);
    return 0;
}
```

Лістінг lab1.cpp

```
#include <windows.h>
#include <iostream>
#include <conio.h>
#include <thread>
#include <chrono>

#define FILE_NAME "data.dat"
#define ARRAY_SIZE 30

// Сортування
void bubbleSort(int* data, int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - 1 - i; j++) {
            if (data[j] > data[j + 1]) {
                std::swap(data[j], data[j + 1]);
            }
        }
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
}
// Очікування натискання пробілу
void waitForSpace() {
    std::cout << "Натисніть ПРОБІЛ для початку сортування..." << std::endl;
    while (true) {
        if (_kbhit() && _getch() == ' ') break;
    }
```

```
}

// М’ютекс
HANDLE initMutex() {
    HANDLE hMutex = OpenMutexA(SYNCHRONIZE, FALSE, "Global\\MyDataMutex");
    if (!hMutex) {
        hMutex = CreateMutexA(NULL, FALSE, "Global\\MyDataMutex");
    }
    return hMutex;
}

int main() {
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);

    waitForSpace();

    // Відкриття файлу
    HANDLE hFile = CreateFileA(FILE_NAME, GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hFile == INVALID_HANDLE_VALUE) {
        std::cerr << "Не вдалося відкрити файл." << std::endl;
        return 1;
    }
    HANDLE hMapping = CreateFileMappingA(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
    if (!hMapping) {
        std::cerr << "Не вдалося створити відображення файлу." << std::endl;
        CloseHandle(hFile);
        return 1;
    }

    int* data = (int*)MapViewOfFile(hMapping, FILE_MAP_ALL_ACCESS, 0, 0, 0);
    if (!data) {
        std::cerr << "Не вдалося відобразити файл у пам’яті." << std::endl;
        CloseHandle(hMapping);
        CloseHandle(hFile);
        return 1;
    }

    HANDLE hMutex = initMutex();
    if (!hMutex) {
        std::cerr << "Не вдалося створити або відкрити м’ютекс." << std::endl;
        UnmapViewOfFile(data);
        CloseHandle(hMapping);
        CloseHandle(hFile);
        return 1;
    }

    std::cout << "Сортування розпочато..." << std::endl;

    if (WaitForSingleObject(hMutex, INFINITE) == WAIT_OBJECT_0) {
        try {
            bubbleSort(data, ARRAY_SIZE);
        }
        catch (...) {
            std::cerr << "Помилка під час сортування!" << std::endl;
        }
        ReleaseMutex(hMutex);
    }
    else {
        std::cerr << "Не вдалося отримати доступ до м’ютексу." << std::endl;
    }

    std::cout << "Сортування завершено." << std::endl;

    UnmapViewOfFile(data);
```

```
    CloseHandle(hMapping);
    CloseHandle(hFile);

    return 0;
}
```

Лістінг view.cpp

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

#define FILE_NAME "data.dat"
#define ARRAY_SIZE 30
#define TIMER_ID 1
#define TIMER_INTERVAL 500

int* data = nullptr;      // Вказівник на відображені дані
HWND hwndGlobal = NULL;

// відображення файлу в пам'ять
bool mapDataFile() {
    HANDLE hFile = CreateFileA(FILE_NAME, GENERIC_READ, FILE_SHARE_READ |
FILE_SHARE_WRITE,
        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hFile == INVALID_HANDLE_VALUE) return false;

    HANDLE hMapping = CreateFileMappingA(hFile, NULL, PAGE_READONLY, 0, 0, NULL);
    if (!hMapping) {
        CloseHandle(hFile);
        return false;
    }

    data = (int*)MapViewOfFile(hMapping, FILE_MAP_READ, 0, 0, 0);
    CloseHandle(hFile);
    CloseHandle(hMapping);

    return data != nullptr;
}
// Вивід зірочок
void drawArrayBars(HDC hdc, RECT& rect) {
    HANDLE hMutex = OpenMutexA(SYNCHRONIZE, FALSE, "Global\\MyDataMutex");
    if (!hMutex) {
        MessageBoxA(NULL, "Неможливо відкрити mutex для читання", "Помилка",
MB_ICONERROR);
        exit(1);
    }
    FillRect(hdc, &rect, (HBRUSH)(COLOR_WINDOW + 1));
    if (WaitForSingleObject(hMutex, 100) == WAIT_OBJECT_0) {
        int y = 10;
        try {
            for (int i = 0; i < ARRAY_SIZE; ++i) {
                int stars = data[i];
                std::string line(stars, '*'); // формуємо рядок зірочок
                TextOutA(hdc, 10, y, line.c_str(), (int)line.length());
                y += 20;
            }
        }
        catch (...) {
            TextOutA(hdc, 10, 10, "Помилка читання з пам'яті", 27);
        }
        ReleaseMutex(hMutex);
    }
    else {
        TextOutA(hdc, 10, 10, "Очікування доступу до даних...", 32);
    }
}
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {
    switch (msg) {
```

```
    case WM_CREATE:
        SetTimer(hwnd, TIMER_ID, TIMER_INTERVAL, NULL);
        return 0;

    case WM_TIMER:
        InvalidateRect(hwnd, NULL, TRUE);
        return 0;

    case WM_PAINT: {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hwnd, &ps);
        drawArrayBars(hdc, ps.rcPaint);
        EndPaint(hwnd, &ps);
        return 0;
    }

    case WM_DESTROY:
        if (data) UnmapViewOfFile(data);
        PostQuitMessage(0);
        return 0;
    }
    return DefWindowProc(hwnd, msg, wParam, lParam);
}

// Точка входу
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, int nCmdShow) {
    if (!mapDataFile()) {
        MessageBoxA(NULL, "Не вдалося відкрити або відобразити файл", "Помилка",
MB_ICONERROR);
        return 1;
    }
    const char CLASS_NAME[] = "DataViewerClass";
    WNDCLASSA wc = {};
    wc.lpfnWndProc = WndProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = CLASS_NAME;
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    RegisterClassA(&wc);
    // Створення вікна
    HWND hwnd = CreateWindowExA(
        0, CLASS_NAME, "Візуалізація даних", WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, CW_USEDEFAULT, 500, 700,
        NULL, NULL, hInstance, NULL);

    if (!hwnd) return 1;
    hwndGlobal = hwnd;
    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);
    // Головний цикл повідомлень
    MSG msg = {};
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```

Висновок: Під час виконання лабораторної роботи було реалізовано програми, які демонструють спільний доступ процесів до одного джерела даних через відображення файлу у пам’ять.