ЛАБОРАТОРНА РОБОТА №1

Тема: Процеси та потоки

Мета роботи: 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.

Додаткове завдання.

Написати четверту програму (консольну), яка буде одночасно працювати, та намагатися відсортувати той самий масив в іншому напрямку та іншим відомим методом сортування.

Листинг програма для створення data.dat файлу:

```
#include <Windows.h>
#include <string>
#include <iostream>
#include <random>

int count = 30;
int min = 10;
int max = 100;

std::string file_name = "data.dat";

int random_integer(int min, int max)
{
   std::random_device rd;
   std::uniform_int_distribution<int> dis(min, max);
   return dis(rd);
}

int main(int argc, char* argv[])
{
   HANDLE h_file = CreateFile(file_name.c_str(), GENERIC_WRITE | GE-
NERIC_READ, FILE_SHARE_READ |
      FILE_SHARE_WRITE, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

   if (h_file == INVALID_HANDLE_VALUE)
   {
      std::cerr << "Error creating the file!" << std::endl;
      return 1;
   }

   int* numbers = new int[count];

   for (int i = 0; i < count; i++)
   {
      numbers[i] = random_integer(min, max);
   }

   DWORD bytes_written;

   BOOL success = WriteFile(h_file, numbers, sizeof(int) * count,
&bytes_written, NULL);
```

```
    if (!success)
    {
        std::cerr << "Error writing to the file!" << std::endl;
        return 1;
    }

    CloseHandle(h_file);

    return 0;
}
```

Програма створює файл data.dat, що міститиме випадкові цілі числа. Вона відкриває файл у двійковому форматі, записуючи у нього 30 випадкових чисел у діапазоні від 10 до 100.

Глобальні зміни:

- std::string file_name – ім'я файлу, у який будуть записані згенеровані числа.
- int count – кількість чисел, що записуються у файл.
- int min– мінімальне значення для випадкових чисел.
- int max – максимальне значення для випадкових чисел.

Методи, використані у програмі:

- Для генерації випадкових чисел використовується функція std::uniform_int_distribution у поєднанні з std::random_device.
- Файл створюється та відкривається за допомогою функції CreateFile, що дозволяє записувати у нього дані у двійковому форматі.
- Перед записом у файл та після нього перевіряється успішність операцій (CreateFile, WriteFile, CloseHandle).

Структура програми:

1. Відкриття файлу для запису (CreateFile).
2. Генерація 30 випадкових чисел у масиві numbers.
3. Запис масиву у файл (WriteFile).
4. Закриття файлу (CloseHandle).

```
PS C:\Users\357\Desktop\SystemNetworkProgramming> run .\setup.cpp
PS C:\Users\357\Desktop\SystemNetworkProgramming> cat .\data.dat
W`b▬▬SPP5*-▼RK∟0
O▼:aR7XR9I

PS C:\Users\357\Desktop\SystemNetworkProgramming>
```

Рис. 1 Робота програми для створення файлу

Лістинг програми для сортування файлу у спадаючому порядку методом вибору:

```
#include <Windows.h>
#include <cstdlib>
#include <fileapi.h>
#include <iostream>
#include <memoryapi.h>
#include <string>
#include <synchapi.h>
#include <winbase.h>
#include <winnt.h>
#include <conio.h>
#include <chrono>
#include <thread>

HANDLE h_mutex;
std::string file_name = "data.dat";

int selection_sort(int* array, int length)
{
   for (int i = 0; i < length - 1; i++)
   {
      int index = i;

      for (int j = i + 1; j < length; j++)
      {
         if (array[index] > array[j])
         {
            index = j;
         }
      }

      if(WaitForSingleObject(h_mutex, INFINITE) == WAIT_OBJECT_0)
      {
         std::swap(array[index], array[i]);
         ReleaseMutex(h_mutex);
      }

      else
      {
         std::cerr << "Error getting a mutex!" << std::endl;
      }

      std::this_thread::sleep_for(std::chrono::milliseconds(100));
   }
```

```
    return 0;
}

int sort_file(std::string file_name)
{
    HANDLE h_file = CreateFile(file_name.c_str(), GENERIC_READ | GENER-
IC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);

    if (h_file == INVALID_HANDLE_VALUE)
    {
        std::cerr << "Error opening the file!" << std::endl;
        return 1;
    }

    HANDLE h_mapping = CreateFileMapping(h_file, NULL, PAGE_READWRITE, 0,
0, NULL);

    if (!h_mapping)
    {
        std::cerr << "Error creating a file mapping object!" << std::endl;
        return 1;
    }

    int* numbers = (int*)MapViewOfFile(h_mapping, FILE_MAP_ALL_ACCESS, 0, 0,
0);

    if (!numbers)
    {
        std::cerr << "Error mapping the file!" << std::endl;
        return 1;
    }

    int file_size = GetFileSize(h_file, NULL);
    int count = file_size / sizeof(int);

    int success = selection_sort(numbers, count);

    UnmapViewOfFile(numbers);
    CloseHandle(h_file);
    CloseHandle(h_mapping);

    return success;
```

```
}

int main(int argc, char* argv[])
{
   h_mutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, "Global\\SortMutex");

   std::cout << "Press space to sort the file." << std::endl;

   while (true)
   {
      char character = _getch();

      if (character == 32)
      {
         std::cout << "Sorting the file..." << std::endl;

         BOOL result = !sort_file(file_name);

         if (result) std::cout << "Array sorted successfully!" << std::endl;
      }

      else if (character == 'q')
      {
         break;
      }
   }

   return 0;
}
```

Програма відкриває файл data.dat, який містить випадкові числа, і сортує їх у зростаючому порядку методом вибору. Щоб уникнути конфліктного доступу до файлу з боку інших програм, застосовується механізм синхронізації потоків (mutex).

Основні параметри:

- std::string file_name – шлях до файлу, в якому зберігаються числа.
- HANDLE h_mutex – дескриптор м'ютекса.

Методи, використані у програмі:

- Сортування вибором – реалізовано у функції selection_sort().
- Відображення файлу в пам’ять – за допомогою CreateFileMapping() і MapViewOfFile(), що дозволяє працювати з даними напряму, без зайвих файлових операцій.
- Синхронізація потоків – виклик WaitForSingleObject(h_mutex, INFINITE)

забезпечує доступ до масиву чисел лише під час обміну елементів.

- Обробка помилок – здійснюється перевірка успішності відкриття файлу, створення його відображення та доступу до пам’яті.
- Алгоритм роботи програми:
- Ініціалізація об'єкта м'ютекса (OpenMutex).
- Відкриття файлу (CreateFile) та створення його відображення у пам’яті (CreateFileMapping, MapViewOfFile).
- Отримання розміру файлу та підрахунок кількості чисел у ньому.
- Очікування натискання клавіші SPACE для запуску сортування.
- Виконання сортування вибором із блокуванням доступу до масиву лише під час запису.
- Звільнення ресурсів: закриття дескрипторів файлу та м'ютекса.

```
PS C:\Users\357\Desktop\SystemNetworkProgramming> run .\sorting.cpp
Press space to sort the file.
Sorting the file...
Array sorted successfully!
```

Рис. 2 Робота для сортування файлу алгоритмом вставки

Лістинг програми для сортування файлу у зростаючому порядку методом бульбашки:

```
#include <Windows.h>
#include <cstdlib>
#include <fileapi.h>
#include <iostream>
#include <memoryapi.h>
#include <string>
#include <synchapi.h>
#include <winbase.h>
#include <winnt.h>
#include <conio.h>
#include <chrono>
#include <thread>

HANDLE h_mutex;
std::string file_name = "data.dat";

int bubble_sort(int* array, int length)
{
    for (int i = 0; i < length - 1; i++)
    {
        for (int j = 0; j < length - 1 - i; j++)
        {
            if (array[j] < array[j + 1])
            {
```

```
            if(WaitForSingleObject(h_mutex, INFINITE) == WAIT_OBJECT_0)
            {
               std::swap(array[j], array[j + 1]);
               ReleaseMutex(h_mutex);
            }

            else
            {
               std::cerr << "Error getting a mutex!" << std::endl;
            }

            std::this_thread::sleep_for(std::chrono::milliseconds(100));
         }
      }
   }

   return 0;
}

int sort_file(std::string file_name)
{
   HANDLE h_file = CreateFile(file_name.c_str(), GENERIC_READ |
GENERIC_WRITE,
      FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);

   if (h_file == INVALID_HANDLE_VALUE)
   {
      std::cerr << "Error opening the file!" << std::endl;
      return 1;
   }

   HANDLE h_mapping = CreateFileMapping(h_file, NULL, PAGE_READWRITE, 0,
0, NULL);

   if (!h_mapping)
   {
      std::cerr << "Error creating a file mapping object!" << std::endl;
      return 1;
   }

   int* numbers = (int*)MapViewOfFile(h_mapping, FILE_MAP_ALL_ACCESS, 0, 0,
0);
```

```
    if (!numbers)
    {
        std::cerr << "Error mapping the file!" << std::endl;
        return 1;
    }

    int file_size = GetFileSize(h_file, NULL);
    int count = file_size / sizeof(int);

    int success = bubble_sort(numbers, count);

    UnmapViewOfFile(numbers);
    CloseHandle(h_file);
    CloseHandle(h_mapping);

    return success;
}

int main(int argc, char* argv[])
{
    h_mutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, "Global\\SortMutex");

    std::cout << "Press space to sort the file." << std::endl;

    while (true)
    {
        char character = _getch();

        if (character == 32)
        {
            std::cout << "Sorting the file..." << std::endl;

            BOOL result = !sort_file(file_name);

            if (result) std::cout << "Array sorted successfully!" << std::endl;
        }

        else if (character == 'q')
        {
            break;
        }
    }

    return 0;
```

```
}
```

Ця програма виконує ту ж саму функцію, що й попередня, але використовує бульбашкове сортування замість сортування вибіркою і сортує числа у спадному порядку, а не у зростаючому.

Лістинг програми для виводу масиву у вікні:

```
#include <Windows.h>
#include <cstring>
#include <fileapi.h>
#include <handleapi.h>
#include <iostream>
#include <memoryapi.h>
#include <minwindef.h>
#include <winnt.h>
#include <winuser.h>
#include "utils.h"

std::string file_name = "data.dat";
char character = '*';
int gap = 20;
int timer_id = 1;
int timer_interval = 500;
HANDLE h_mutex;

std::string get_characters(char character, int count)
{
   std::string result = "";

   for (int i = 0; i < count; i++)
   {
      result = result + character;
   }

   return result;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
   switch (uMsg)
   {
      case WM_CREATE:
      {
         return 0;
```

```
        }

        case WM_PAINT:
        {
            PAINTSTRUCT ps;

            HANDLE h_file = CreateFile(file_name.c_str(), GENERIC_READ |
GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);

            if (h_file == INVALID_HANDLE_VALUE)
            {
                std::cerr << "Error reading the file!" << std::endl;
                return 1;
            }

            int file_size = GetFileSize(h_file, NULL);

            HANDLE h_mapping = CreateFileMapping(h_file, NULL,
PAGE_READWRITE, 0, 0, NULL);

            if (h_mapping == NULL)
            {
                std::cerr << "Error creating a file mapping object!" << std::endl;
                return 1;
            }

            int* numbers = (int*)MapViewOfFile(h_mapping, FILE_MAP_ALL_ACCESS,
0, 0, 0);

            if (!numbers)
            {
                std::cerr << "Error mapping the file!" << std::endl;
                return 1;
            }

            int count = file_size / sizeof(int);

            HDC hdc = BeginPaint(hwnd, &ps);

            for (int i = 0; i < count; i++)
            {
                TextOut(hdc, 0, i * gap, get_characters(character, numbers[i]).c_str(),
```

```
numbers[i]);
            }

            CloseHandle(h_file);
            CloseHandle(h_mapping);

            SetTimer(hwnd, timer_id, timer_interval, NULL);
            EndPaint(hwnd, &ps);

            return 0;
        }

        case WM_TIMER:
        {
            if (wParam == timer_id)
            {
                InvalidateRect(hwnd, NULL, TRUE);
            }

            return 0;
        }

        case WM_DESTROY:
        {
            CloseHandle(h_mutex);
            PostQuitMessage(0);
            return 0;
        }
    }

    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

int main()
{
    h_mutex = CreateMutex(NULL, FALSE, "Global\\SortMutex");

    if (h_mutex == NULL)
    {
        std::cerr << "Error creating a mutex!" << std::endl;
    }

    HINSTANCE hInstance = GetModuleHandle(NULL);
```

```
WNDCLASS wc = {0};

wc.lpfnWndProc = WindowProc;
wc.hInstance = hInstance;
wc.lpszClassName = "SomeWindow";
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);

if (!RegisterClass(&wc))
{
   return 1;
}

HWND hwnd = CreateWindowEx(
   0,
   wc.lpszClassName,
   "Window",
   WS_OVERLAPPEDWINDOW,
   CW_USEDEFAULT,
   CW_USEDEFAULT,
   500,
   300,
   NULL,
   NULL,
   hInstance,
   NULL
);

if (hwnd == NULL)
{
   std::cerr << "Error creating the window!" << std::endl;
   return 1;
}

ShowWindow(hwnd, SW_SHOW);
UpdateWindow(hwnd);

MSG msg = {0};

while (GetMessage(&msg, NULL, 0, 0))
{
   TranslateMessage(&msg);
   DispatchMessage(&msg);
}
```

```
    return 0;
}
```

Програма відкриває файл data.dat, який містить випадкові числа, та відображає їх у графічному вікні у вигляді рядків із заданим символом. Кількість символів у кожному рядку відповідає значенню числа у файлі. Оновлення відображення здійснюється періодично за допомогою таймера.

Глобальні змінні:

- std::string file_name = "data.dat"; – Зберігає ім'я файлу (data.dat), що містить числа.
- char character = '*'; – Визначає символ, який використовується для відображення чисел у вікні.
- int gap = 20; – Визначає вертикальний проміжок між рядками при відображенні чисел.
- int timer_id = 1; – Зберігає ідентифікатор таймера для періодичних оновлень вікна.
- int timer_interval = 500; – Визначає інтервал (в мілісекундах) для таймера.
- HANDLE h_mutex; – Містить дескриптор м'ютекса.

Структура програми:

1. Створення та ініціалізація mutex (CreateMutex).
2. Реєстрація класу вікна та створення вікна (RegisterClass, CreateWindowEx).
3. Відкриття файлу та створення його відображення у пам'яті (CreateFile, CreateFileMapping, MapViewOfFile).
4. Визначення розміру файлу та кількості чисел у ньому.
5. Відображення вмісту файлу у вікні та відновлення кожні 500 мс через таймер.
6. Завершення програми: закриття ресурсу mutex, закриття дескрипторів файлу та м'ютекса.

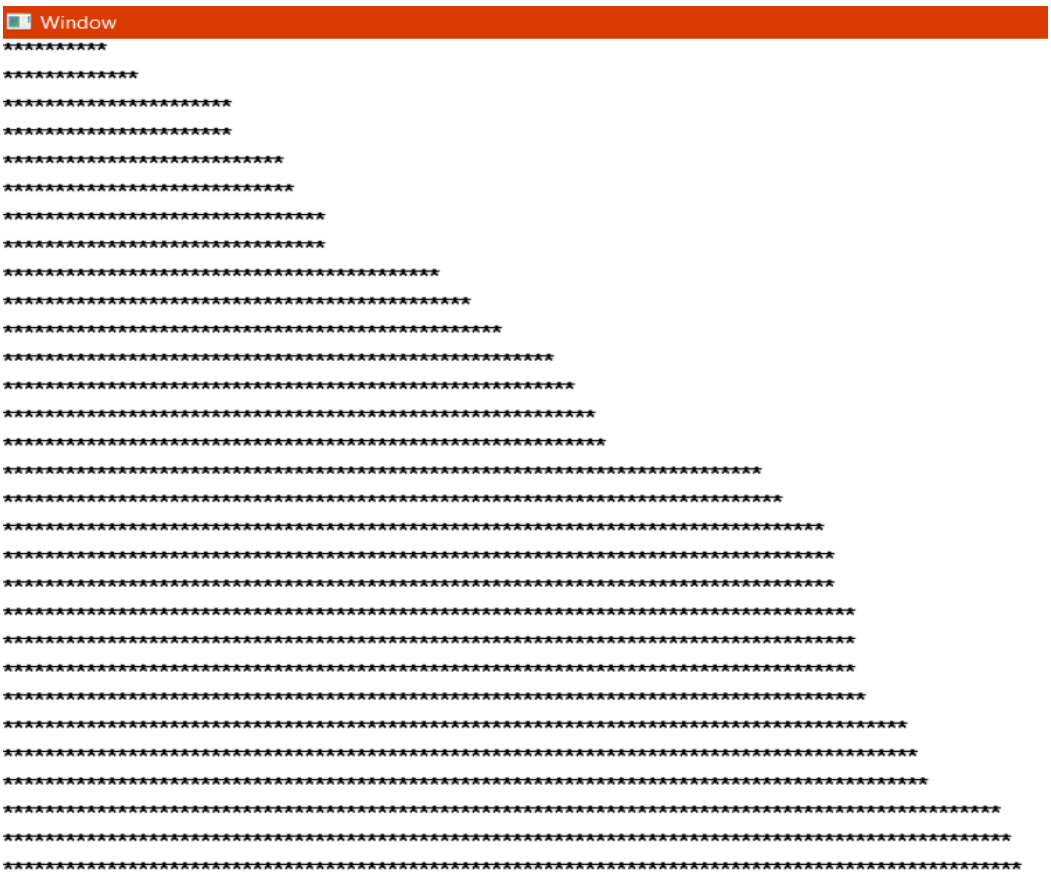

Рис. 3 Зовнішній вигляд вікна

Посилання на репозиторій: https://git.ztu.edu.ua/ipz231_soa/system-network-programming/-/tree/lab1

Висновок: У результаті виконання лабораторної роботи було продемонстровано ефективну взаємодію між процесами через спільний доступ до даних за допомогою проектування файлів в пам'ять. Завдяки застосуванню механізмів синхронізації, таких як mutex або semaphor, забезпечено коректну роботу з даними та обробку виняткових ситуацій, що дозволяє уникнути помилок при паралельному доступі до спільних ресурсів.