# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**ЗВІТ**

**Про виконання лабораторної роботи №4**

**«Синхронізація потоків 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**: Синхронізація потоків в ОС Windows.

**Мета роботи**: Навчитися організовувати багатопоточність з використанням синхронізації за допомогою функцій WinAPI.

## 1. Теоретичні відомості

При роботі з потоками необхідно мати можливість координувати їхні дії. Часто така координація має на меті певний порядок виконання операцій. Крім функцій, призначених для створення потоків та зміни їх планового пріоритету, Win32 API містить функції, які переводять потоки в режим очікування сигналів від певних об'єктів, наприклад від файлів або процесів. Крім цього, ці функції забезпечують підтримку деяких спеціальних об'єктів, зокрема семафорів і виключних семафорів (мютексів).

Найкраще проілюструвати застосування синхронізуючих об'єктів на прикладі функцій, які очікують сигналу від об'єкта **(wait-функцій)**. За допомогою одного набору узагальнених команд можна організувати очікування сигналів від процесів семафорів, виключних семафорів (мютексів) і деяких інших об'єктів. Наступна функція очікує надходження сигналу від зазначеного об'єкту:

*DWORD WaitForSingleObject*

*(HANDLE hObject, // ​​об'єкт, сигнал від якого очікується DWORD dwMilliseconds // ​​максимальний час очікування);*

Функція *WaitForSingleObject()* дозволяє призупинити виконання потоку до тих пір, поки не надійде сигнал від заданого об'єкта. Крім того, в цій команді вказується максимальний час очікування. Щоб забезпечити нескінченне очікування, в якості тимчасового інтервалу слід задати значення *INFINITE*.

Для того щоб змусити потік очікувати сигнали відразу від декількох об'єктів, використовують функцією *WaitForMultipleObjects().* Функція повертає управління потоку при надходженні сигналу або від одного із зазначених об'єктів, або від всіх об'єктів разом. У програмі, керованої події, повинен бути заданий масив об'єктів:

*DWORD WaitForMultipleObjects*

*( DWORD dwNumObjects , // ​​кількість очікуваних об'єктів*

*LPHAMDLE lpHandles , // ​​масив дескрипторів*

*BOOL bWaitAll , // ​​TRUE - очікування сигналів відразу від всіх об'єктів;*

*// FALSE - очікування сигналу від будь-якого з об'єктів*

*DWORD dwMilliseconds // ​​максимальний період очікування);*

Функціям, що створюють **семафори** і **мютекси**, потрібно вказувати необхідні привілеї доступу і початкові параметри створюваного об'єкта (можна також вказати його ім'я , але це необов'язково).

*HANDLE CreateMutext*

*(LPSECURITY\_ATTRIBUTES lpsa ,*

*// Необов'язкові атрибути безпеки*

*BOOL bInitialOwner , // ​​TRUE - творець хоче*

*// заволодіти отриманим об'єктом*

*LPTSTR lpszMutexName // ​​ім'я об'єкта);*

*HANDLE CreateSemaphore*

*( LPSECURITY\_ATTRIBUTES lpsa ,*

*// необов'язкові атрибути безпеки*

*LONG UnitialCount, // ​​початкове значення лічильника (зазвичай 0)*

*LONG lMaxCount, // ​​максимальне значення*

                    // лічильника (обмежує число потоків)

LPTSTR lpszSemName // ​​ім'я семафора (може мати значення NULL);

Поки семафор (мютекс) існує, потік взаємодіє з ним за допомогою операцій захоплення і звільнення. Для захоплення будь-якого об'єкта потік викликає функцію *WaitForSingleObject* (або одну з її різновидів). Завершивши виконання завдання, яка синхронізувалася захопленим об'єктом, потік звільняє цей об'єкт за допомогою однієї з таких функцій:

*BOOL ReleaseMutex (HANDLE hMutex);*

*BOOL ReleaseSemaphore*

*(HANDLE hSemaphore ,*

*LONG lRelease , // ​​величина, на яку інкрементується*

*// значення лічильника при звільненні об'єкта (зазвичай 1)*

*LPLONG lplPrevious, // ​​змінна, якій присвоюється*

*//попереднє значення лічильника);*

Сімейство **Interlocked-функцій** дає нам ключ до вирішення проблеми, коли потрібно змінити значення глобальної змінної в декількох потоках. Наприклад функція:

*LONG InterlockedExchangeAdd (PLONG plAddend, LONG lIncrement);*

Ми викликаємо цю функцію, передаючи адресу змінної типу *LONG* і вказуємо додане значення. *InterlockedExchangeAdd* гарантує, що операція буде виконана атомарно.

Ось приклад фрагменту програми:

*// Визначаємо глобальну змінну long g\_x = 0 ;*

*DWORD WINAPI ThreadFunc1 ( PVOID pvParam ) {*

*InterlockedExchangeAdd ( & g\_x , 1 ) ;*

*return ( 0 ) ;*

*}*

*DWORD WINAPI ThreadFunc2 ( PVOID pvPararr ) {*

*InterlockedExchangeAdd ( & g\_x , 1 ) ;*

*return ( 0 ) ;*

*}*

Ось ще дві функції з цього сімейства:

*LONG InterlockedExchange (PLONG plTarget , LONG IValue) ;*

*PVOTD InterlockedExchangePointer (PVOID \* ppvTarget , PVOID \* pvValue );*

*InterlockedExchange* і *InterlockedExchangePointer* монопольно замінюють поточне значення змінної типу *LONG* , адреса якої передається в першому параметрі, на значення, що передається в другому параметрі. Ці функції повертають початкове значення змінної. *InterlockedExchange* надзвичайно корисна при реалізації **спін-блокування (spinlock):**

*// Глобальна змінна , використовувана як індикатор того, чи зайнятий спільний ресурс*

*BOOL g\_fResourceInUse = FALSE ;*

*...*

*void Func1 () {*

*// Очікуємо доступу до ресурсу*

*while ( InterlockedExchange ( & g\_fResourceInUse , TRUE ) = TRUE )*

*Sleep (0);*

*// Отримуємо ресурс у своє розпорядження*

*...*

*// Доступ до ресурсу більше не потрібен*

*InterlockedFxchange ( & g\_fResourceInUse , FALSE ) ;}*

**Критична секція (critical section)** - це невелика ділянка коду, що вимагає монопольного доступу до якихось загальних даних. Вона дозволяє зробити так, щоб одноразово тільки один потік отримував доступ до певного ресурсу. Зрозуміло, що система може в будь-який момент витіснити Ваш потік і підключити до процесу інший, але жоден з потоків, яким потрібен зайнятий Вами ресурс, по одержить процесорний час до тих пір, поки Ваш потік не вийде за межі критичної секції.

## 2. Хід роботи

## Завдання (варіант №5).

1. Реалізувати алгоритм з лабораторної роботи №3: створити масив N елементів і відсортувати його елементи у порядку зростання за допомогою методу «вибірки» (елементи масиву згенерувати випадковим чином за допомогою вбудованих функцій).
2. Виконати розпаралелювання заданого алгоритму на 2, 4, 8 потоки із використанням синхронізації.
3. Реалізувати прогрес (хід) виконання задачі.
4. Для синхронізації потоків використати такі методи: Interlocked-функції, мютекси, семафори, критична секція, спін-блокування, Wait-функції.
5. Результати виконання роботи відобразити у звіті.
6. Створимо потік в функції *main()* нашої програми наступним чином:

*int main() {*

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

*HANDLE hThread[K]; //масив дескрипторів потоків*

*DWORD ThreadId[K]; //унікальні ідентифікатори потоків*

*//створюємо потоки*

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

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

*return 0;*

*}*

Функція, в якій буде виконуватись потік *Output0(…)* матиме вигляд:

*//функція опрацювання 1-го потоку*

*DWORD WINAPI Output0(LPVOID Param) {*

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

*return 0;*

*}*

У функції *sort(…) -* буде виконуватись алгоритм сортування, що має вигляд:

*//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);*

*}*

1. Для створення декількох потоків, використаємо наступний код:

*const int K = 8; //к-сть потоків*

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

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

Опрацювання цих потоків будемо здійснювати у функціях *Output1… Output7*, аналогічно як у п.1.

Поділ вхідного масиву на під масиви реалізуємо наступним чином:

*//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;*

*}*

*}*

1. Для реалізації прогресу створимо функцію *progress()*, яка буде викликатись з функції *sort()*:

*//функція виводу прогресу на екран*

*void progress(int value) {*

*system("cls");*

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

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

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

*}*

1. Реалізовуємо **спін-блокування**, а також **interlocked-функції** наступним чином:

*//"прапорець" для спін-блокування*

*LONG resourceInUse = FALSE;*

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

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

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

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

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

*}*

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

*}*

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

*//Interlocked-інкремент*

*InterlockedIncrement(&threadFinalsIndex);*

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

*}*

**Критичні секції** використовуємо так:

*//критичні секції*

*CRITICAL\_SECTION cs, cs2;*

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

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

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

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

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

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

*}*

*// 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);*

*}*

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

*int main() {*

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

*//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);*

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

*return 0;*

*}*

**Семафори, мютекси та wait-функції** використовуємо для синхронізації присвоєння значень змінних та виводу їх на екран:

*//дескриптор семафора*

*HANDLE hSemaphore;*

*//дескриптор мютекса*

*HANDLE hMutex;*

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

*//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;*

*}*

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

*int main() {*

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

*//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()*

*printf("\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*

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

*}*

*//закриваємо дескриптори семафору та потоку*

*CloseHandle(hSemaphore);*

*CloseHandle(hInitThread);*

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

*//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);*

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

*return 0;*

*}*

1. Кінцевий текст програми міститься в розділі 3, а результати виконання див. розділ 4.

## Текст програми

#include <iostream>

#include <cstdlib>

#include <stdio.h>

#include <conio.h>

#include <math.h>

#include <time.h>

#include <windows.h>

using namespace std;

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

//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) {

system("cls");

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

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

printf("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() {

setlocale(LC\_ALL,"Ukrainian");

//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()

printf("\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

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

}

//закриваємо дескриптори семафору та потоку

CloseHandle(hSemaphore);

CloseHandle(hInitThread);

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

\_getch();

//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]);

}

//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();

return 0;

}

## Результат виконання програми

Запустимо програму на виконання з ***одним потоком*** (рис. 4.1 - 4.2).

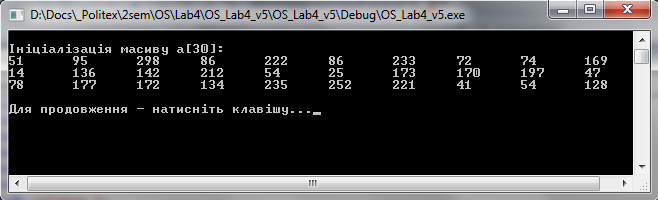


Рис. 4.1. Один потік (до сортування)

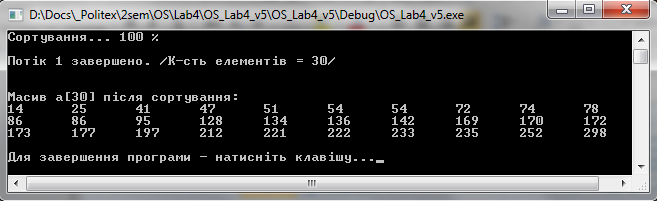


Рис. 4.2. Один потік (після сортування)

Збільшимо ***кількість потоків до двох*** та запустимо програму (рис. 4.3).

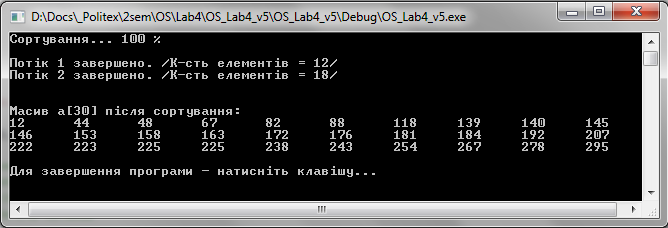


Рис. 4.3. Два потоки

Як бачимо, перший потік завершився швидше ніж другий, оскільки в ньому опрацьовується менша кількість елементів і він стартує першим.

Збільшимо ***кількість потоків до чотирьох*** (рис. 4.4).

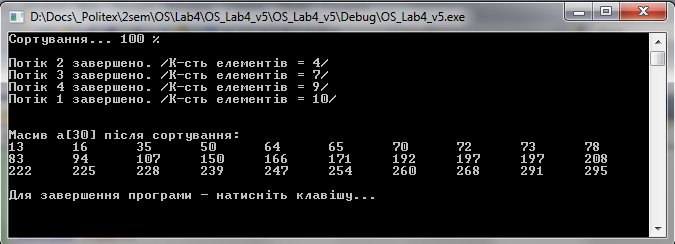


Рис. 4.4. Чотири потоки

Як бачимо, завершуютья швидше ті потоки, які опрацьовують найменшу кількість елементів.

Збільшимо ***кількість потоків до вісьмох*** (рис. 4.5).

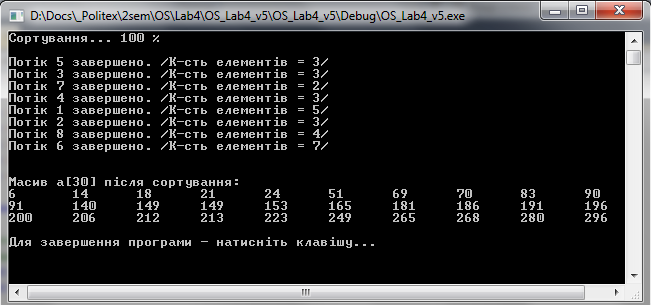


Рис. 4.5. Вісім потоків

В даному випадку бачимо, що тенденція швидшого завершення потоків з найменшою кількістю елементів масиву, справджується не завжди. Це можна пояснити, тим що в даному прикладі, розподіл кількостей між потоками вийшов досить близьким до рівномірного, що і підвищує шанси завершитись швидше потокам, які, в свою чергу, швидше стартували.

## ВИСНОВКИ

На даній лабораторній роботі я ознайомився з багатопоточністю в ОС Windows. Навчився працювати з потоками, використовуючи WinAPI-функції, зокрема запускати декілька потоків, призупиняти, відновлювати їхню роботу та закривати їх, також реалізовувати розпаралелювання алгоритмів з синхронізацією, зокрема для синхронізації потоків використовувати такі методи, як: Interlocked-функції, мютекси, семафори, критичні секції, спін-блокування, Wait-функції.