# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**ЗВІТ**

**Про виконання лабораторної роботи №3**

**«Виконання задачі в декількох потоках 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**: Виконання задачі в декількох потоках в ОС Windows.

**Мета роботи**: Навчитись реалізовувати розпаралелювання алгоритмів за допомогою багатопоточності в ОС Windows з використанням функцій WinAPI.

## 1. Теоретичні відомості

Сортування вибором (англійською «Selection Sort») — простий алгоритм сортування лінійного масиву, на основі вставок. Має ефективність O(n2), що робить його неефективним при сортування велеких масивів, і в цілому, менш ефективним за подібний алогоримт сортування включенням. Сортування вибором вирізняється більшою простотою, ніж cортування включенням, і в деяких випадках вищою продуктивністю.

Алгоритм працює наступним чином:

1. Знаходить у списку найменше значення.

2. Міняє його місцями із першим значеннями у списку.

3. Повторює два попередніх кроки, доки список не завершиться (починаючи з другої позиції).

Фактично, таким чином ми поділили список на дві частини: перша (ліва) — повністю відсортована, а друга (права) — ні.

Сортування вибором не є складним в аналізі та порівнянні його з іншими алгоритмами, оскільки жоден з циклів не залежить від даних у списку. Знаходження найменшого елементу вимагає перегляду усіх n елементів (у даному випадку (n − 1) порівняння), і після цього, переситановки його до першої позиції. Знаходження наступного найменшого елементу вимагає перегляду (n − 1) елементів, і так далі, для (n − 1) + (n − 2) + ... + 2 + 1 = n(n − 1) / 2 Î Θ(n2) порівнянь. Кожне сканування вимагає однієї перестановки для (n − 1) елементів (останній елемент знаходитиметься на своєму місці).

Реалізація алгоритму ***/алгоритм S /*:**

Задана таблиця елементів *R1 ,R2 ,…,Rn.*

Даний алгоритм реорганізує таблицю у висхідному порядку, тобто для її ключів буде мати місце співвідношення k*1 <k2 <…<kn.*

**Алгоритм S.**

S1. Цикл по індексу проходження. Повторювати кроки S2 - S4 при i=1,2,…,n-1.

S2. Ініціалізація мінімального індексу пошуку: встановити r=i.

S3. Відшукання найменшого ключа. Повторювати при j=i+1*,*i+2,…,n:якщо k*j <kr* то встановити r = j*.*

S4. Перестановка записів. Якщо r !=j, то R*i<->Rr.*

S5. Кінець, Вихід.

Для того, щоб розпаралелити алгоритм сортування вибором, потрібно спочатку розбити вхідний масив на підмасиви в залежності від величин значень їх елементів, а потім кожен підмасив відсортувати за допомогою прямого алгоритму. Наприклад, якщо на вході у нас є масив зі значеннями елементів від 10 до 70 - *A(10≤ai≤70)* і його сортування потрібно провести в трьох потоках паралельно: спочатку ділимо його на підмасиви *B(10≤bi<30), C(30≤ci<50), D(50≤di≤70),* а потім сортуємо кожен в окремому потоці. Таким чином отримаємо вихідний посортований масив *A = {B, C, D}*.

## 2. Хід роботи

## Завдання (варіант №5).

1. Реалізувати заданий алгоритм в окремому потоці. 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 потоків.
3. Реалізувати можливість зміни пріоритету певного потоку.
4. Результати виконання роботи відобразити у звіті.
5. 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;*

*}*

*}*

*// Міняємо поточний елемент з мінімальним*

*int temp = arr[i];*

*arr[i] = arr[minValueIndex];*

*arr[minValueIndex] = temp;*

*}*

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

*}*

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. Запуск та зміну пріоритетів потоків, реалізуємо так:

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

*printf("Потiк %d:\n", i+1);*

*show(b[i], size[i]);*

*printf("\n");*

*//Встановлюємо пріоритет потоків:*

*//не парним - високий пріоритет, парним - низький*

*SetThreadPriority(hThread[i],i%2 ?THREAD\_PRIORITY\_HIGHEST:THREAD\_PRIORITY\_LOWEST);*

*//запуск потоків*

*ResumeThread(hThread[i]);*

*}*

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

## Текст програми

#include <iostream>

#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 = 2; //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

int threadFinalsIndex = 0;

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

//вивід одновимірного масиву на екран

void show(int arr[], int n) {

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

printf("%d\t", arr[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;

}

}

// Міняємо поточний елемент з мінімальним

int temp = arr[i];

arr[i] = arr[minValueIndex];

arr[minValueIndex] = temp;

}

//Зберігаємо індекс потоку, який завершився

//в глобальний масив

threadFinals[threadFinalsIndex++] = threadNum;

}

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

//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");

//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);

printf("Масив a[%d] до сортування:\n", N);

show(a, N);

//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;

}

}

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]);

printf("\nМасиви до сортування в потоках:\n");

for(int i = 0; i < K; i++){

printf("Потiк %d:\n", i+1);

show(b[i], size[i]);

printf("\n");

//Встановлюємо пріоритет потоків:

//не парним - високий пріоритет, парним - низький

SetThreadPriority(hThread[i], i%2 ? THREAD\_PRIORITY\_HIGHEST : THREAD\_PRIORITY\_LOWEST);

//запуск потоків

ResumeThread(hThread[i]);

}

//чекаємо на завершення всіх потоків

WaitForMultipleObjects(K, hThread, TRUE,INFINITE);

//завершуємо всі потоки в порядку їх завершення

DWORD ExitCode[K];

for(int j = 0; j < K; j++) {

int i = threadFinals[j];

printf("\nСортування в потоцi %d завершено... Прiоритет потоку: %d", i+1, GetThreadPriority(hThread[i]));

//отримуємо код завершення потоків

GetExitCodeThread(hThread[i], &ExitCode[i]);

//завершення потоку

TerminateThread(hThread[i], ExitCode[i]);

//закриття дексриптора потоку

CloseHandle(hThread[i]);

}

printf("\n\nМасив a[%d] пiсля сортування:\n", N);

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

show(b[i], size[i]);

\_getch();

return 0;

}

## Результат виконання програми

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

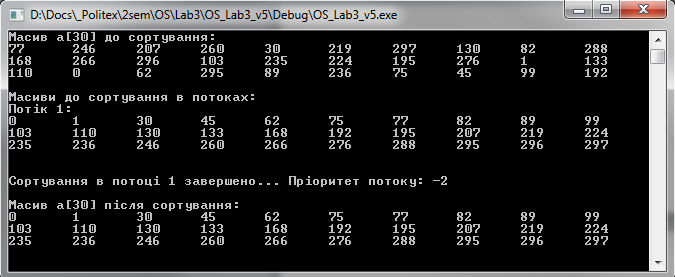


Рис. 4.1. Один потік

Збільшимо ***кількість потоків до двох*** та запустимо програму (рис. 4.2).

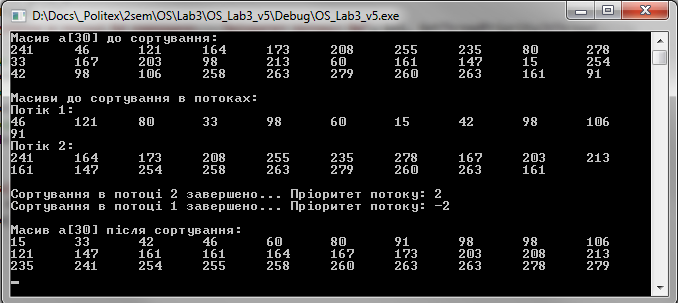


Рис. 4.2. Два потоки

Як бачимо, другий потік завершився швидше ніж перший, оскільки в нього вищий пріоритет.

Збільшимо ***кількість потоків до чотирьох*** (рис. 4.3).

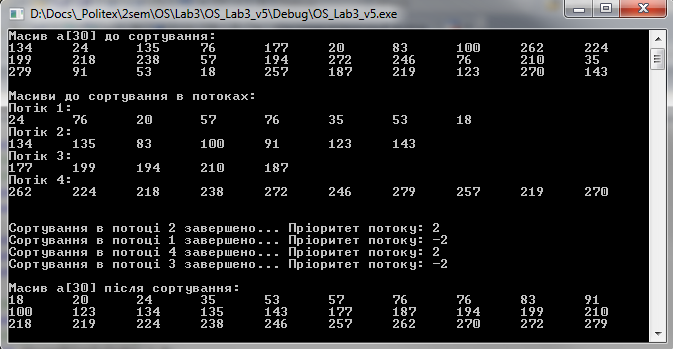


Рис. 4.3. Чотири потоки

Як бачимо, другий потік завершився швидше ніж перший, а четвертий швидше ніж третій, по тим самим причинам що й у попередньому випадку.

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

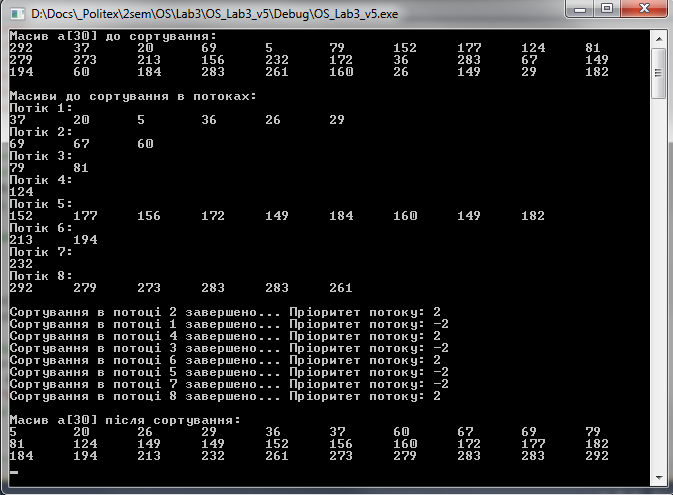


Рис. 4.4. Вісім потоків

В даному випадку бачимо, що тенденція швидшого завершення потоків з вищим пріоритетом збереглася, окрім випадку з сьомим та восьмим потоками. Це можна пояснити, тим що кількість елементів масиву у сьомому потоці лише 1, а у восьмому -6.

## 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 за допомогою багатопоточності в ОС Windows з використанням функцій WinAPI.