Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра СШІ



**Звіт**

до лабораторної роботи № 3

з дисципліни

​*Операційні системи* ​

на тему:

“ Виконання задачі в декількох потоках в ОС Windows ”

Виконав: студент КН-217

Мельничук Борис

Прийняв: Дмитро Петренко

Львів – 2022

**Тема.** Виконання задачі в декількох потоках в ОС Windows

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

**Завдання**.

1. Підготувати файли з тестовими даними різного розміру: від малих прикладів для відлагоджування – і до десятків мегабайт для тестування швидкодії.

Тестові дані згенерувати випадковим чином за допомогою вбудованих функцій, а у випадку роботи зі зображенями – вручну.

1. Вирішити поставлену задачу в окремому потоці. Результат зберегти у файл.
2. Виконати розпаралелювання заданого алгоритму на 2, 4, 8, 20, 100, 1000 потоків.   
   В реалізації слід обійтись БЕЗ використання синхронізаційних конструкцій. Переконатись, що результат співпадає із одно-поточною імплементацією.  
   Проаналізувати вплив розпаралелювання задачі на час виконання.
3. [Окрім задач, у яких потрібно працювати зі словами та послідовностями] Модифікуйте програму так, щоб потоки працювали не із послідовними секціями даних (наприклад, 1й потік – елементи масиву 0..500 000, а 2й потік – 500 001..1000 000), а змішано (1й потік – парні елементи, а 2й потік – непарні). Проведіть аналогічні тести із тими ж кількостями потоків, що й у попередньому пункті. Проаналізуйте вплив «перестановки доданків» на сумарний час виконання та поясніть результати.
4. Реалізувати можливість зміни пріоритету окремого потоку та проаналізувати вплив на тривалість виконання його частини задачі.
5. Результати виконання роботи відобразити у звіті.
   1. Текст програми-генератора тестових даних.
   2. Текст програми-рішення задачі.

Якщо рішення містить блоки запозиченого коду (stackoverflow та ін.) – виділити його курсивом та у коментарях вказати джерело.

Розуміння скопійованого коду – обов’язкове!

* 1. Вхідні-вихідні дані (для великих файлів – зразки та посилання на повні файли).
  2. Час виконання в залежності від кількості потоків при розбитті задачі на суміжні інтервали. Пояснення часових характеристик.
  3. Час виконання в залежності від кількості потоків при розбитті задачі на не-суміжні елементи у межах кожного потоку (парні – 1й потік / непарні – 2й потік). Пояснення часових характеристик.
  4. Час виконання окремого потоку із пониженим/підвищеним пріоритетом у порівнянні із його «братами» стандартного пріоритету. Пояснення результату.

Варіант 23

a)

void generate\_file(ll n, ll m, ll d, ll e)

{

ofstream file(to\_string(n) + "x" + to\_string(m) + "\_" + to\_string(d) + "x" + to\_string(e) + ".txt");

file << n << " " << m << endl;

ll k = 0;

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

{

for (ll j = 0; j < m; j++)

{

k++;

if (j == m - 1)

{

file << k;

}

else

{

file << k << " ";

}

}

file << endl;

}

file << d << " " << e;

k = 0;

for (ll i = 0; i < d; i++)

{

for (ll j = 0; j < e; j++)

{

k++;

if (j == e - 1)

{

file << k;

}

else

{

file << k << " ";

}

}

file << endl;

}

file.close();

}

b)

#include <iostream>

#include <thread>

#include <vector>

#include <fstream>

#include <chrono>

#include <omp.h>

#include <string>

//#include <Windows.h>

#define MAX\_THREADS 1000

using namespace std;

typedef long long int ll;

typedef struct MyData {

vector<vector<ll>> a, b, c;

vector<pair<int, int>> indexes;

} MYDATA, \*PMYDATA;

void generate\_file(ll n, ll m, ll d, ll e)

{

ofstream file(to\_string(n) + "x" + to\_string(m) + "\_" + to\_string(d) + "x" + to\_string(e) + ".txt");

file << n << " " << m << endl;

ll k = 0;

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

{

for (ll j = 0; j < m; j++)

{

k++;

if (j == m - 1)

{

file << k;

}

else

{

file << k << " ";

}

}

file << endl;

}

file << d << " " << e;

k = 0;

for (ll i = 0; i < d; i++)

{

for (ll j = 0; j < e; j++)

{

k++;

if (j == e - 1)

{

file << k;

}

else

{

file << k << " ";

}

}

file << endl;

}

file.close();

}

vector<vector<ll>> matrix\_multiplication(vector<vector<ll>> a, vector<vector<ll>> b)

{

// a (n, m), b (d, e) => res (n, e)

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

ll n = a.size();

ll m = a[0].size();

ll e = b[0].size();

vector<vector<ll>> res(n, vector<ll>(e, 0));

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

{

for (ll j = 0; j < e; j++)

{

for (ll k = 0; k < m; k++)

{

res[i][j] += a[i][k] \* b[k][j];

}

}

}

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::milliseconds>(stop - start);

std::cout << "Time taken by function matrix\_multiplication: " << duration.count() << " milliseconds" << endl;

return res;

}

void vector\_multiplication(vector<vector<ll>> a, vector<vector<ll>> b, vector<vector<ll>> &c, vector<pair<int, int>> indexes)

{

cout << "thread id: " << this\_thread::get\_id() << endl;

for (int idx = 0; idx < indexes.size(); idx++)

{

for (int k = 0; k < b.size(); k++)

{

int x = indexes[idx].first;

int y = indexes[idx].second;

//cout << indexes[idx].first << " " << indexes[idx].second << endl;

c[x][y] += a[x][k] \* b[k][y];

}

}

}

//DWORD vector\_multiplication(LPVOID lpParam)

//{

// PMYDATA pDataArray;

// pDataArray = (PMYDATA)lpParam;

// //cout << "thread id: " << this\_thread::get\_id() << endl;

// for (int idx = 0; idx < pDataArray->indexes.size(); idx++)

// {

// for (int k = 0; k < pDataArray->b.size(); k++)

// {

// int x = pDataArray->indexes[idx].first;

// int y = pDataArray->indexes[idx].second;

// //cout << indexes[idx].first << " " << indexes[idx].second << endl;

// pDataArray->c[x][y] += pDataArray->a[x][k] \* pDataArray->b[k][y];

// }

// }

// return 0;

//}

vector<vector<ll>> matrix\_multiplication\_parallel(vector<vector<ll>> a, vector<vector<ll>> b, ll NUM\_THREADS, vector<vector<pair<int, int>>> indexes)

{

// a (n, m), b (d, e) => res (n, e)

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

ll n = a.size();

ll m = a[0].size();

ll e = b[0].size();

if (NUM\_THREADS > n \* e)

{

cout << "Too many threads" << endl;

NUM\_THREADS = n \* e;

}

vector<thread> th(NUM\_THREADS);

vector<vector<ll>> c(n, vector<ll>(e, 0));

/\*for (lli i = 0; i < n; i++)

{

for (lli j = 0; j < e; j++)

{

for (lli k = 0; k < m; k++)

{

res[i][j] += a[i][k] \* b[k][j];

}

}

}\*/

for (int i = 0; i < NUM\_THREADS; i++) th[i] = thread(vector\_multiplication, a, b, ref(c), indexes[i]);

for (int i = 0; i < NUM\_THREADS; i++) th[i].join();

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::milliseconds>(stop - start);

cout << "Time taken by function: matrix\_multiplication\_parallel " << duration.count() << " milliseconds" << endl;

return c;

}

//vector<vector<ll>> matrix\_multiplication\_parallel\_winapi(vector<vector<ll>> a, vector<vector<ll>> b, ll NUM\_THREADS, vector<vector<pair<int, int>>> indexes)

//{

// // a (n, m), b (d, e) => res (n, e)

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

// ll n = a.size();

// ll m = a[0].size();

// ll e = b[0].size();

// if (NUM\_THREADS > n \* e)

// {

// std::cout << "Too many threads" << endl;

// NUM\_THREADS = n \* e;

// }

// PMYDATA pDataArray[MAX\_THREADS];

// HANDLE th[MAX\_THREADS];

// vector<DWORD> dwThreadIdArray(NUM\_THREADS);

// vector<vector<ll>> c(n, vector<ll>(e, 0));

// /\*for (lli i = 0; i < n; i++)

// {

// for (lli j = 0; j < e; j++)

// {

// for (lli k = 0; k < m; k++)

// {

// res[i][j] += a[i][k] \* b[k][j];

// }

// }

// }\*/

// for (int i = 0; i < NUM\_THREADS; i++)

// {

// pDataArray[i] = (PMYDATA)HeapAlloc(GetProcessHeap(), HEAP\_ZERO\_MEMORY, sizeof(MYDATA));

//

// if (pDataArray[i] == NULL)

// {

// std::cout << "exit" << endl;

// ExitProcess(2);

// }

// std::cout << i << endl;

//

// pDataArray[i]->a = a;

// pDataArray[i]->b = b;

// pDataArray[i]->c = c;

// pDataArray[i]->indexes = indexes[i];

//

//

// th[i] = CreateThread(

// NULL, // default security attributes

// 0, // use default stack size

// vector\_multiplication, // thread function name

// pDataArray[i], // argument to thread function

// 0, // use default creation flags

// &dwThreadIdArray[i]); // returns the thread identifier

//

// if (th[i] == NULL)

// {

// std::cout << "couldn't create a thread" << endl;

// ExitProcess(3);

// }

// }

// //for (int i = 0; i < NUM\_THREADS; i++) th[i].join();

//

// WaitForMultipleObjects(NUM\_THREADS, th, TRUE, INFINITE);

//

// for (int i = 0; i < NUM\_THREADS; i++)

// {

// CloseHandle(th[i]);

// if (th[i] != NULL)

// {

// HeapFree(GetProcessHeap(), 0, th[i]);

// th[i] = NULL; // Ensure address is not reused.

// }

// }

//

// auto stop = chrono::high\_resolution\_clock::now();

// auto duration = chrono::duration\_cast<chrono::microseconds>(stop - start);

// std::cout << "Time taken by function: matrix\_multiplication\_parallel " << duration.count() << " microseconds" << endl;

// return c;

//}

void print\_matrix(vector<vector<ll>> a)

{

for (ll i = 0; i < a.size(); i++)

{

for (ll j = 0; j < a[0].size(); j++) std::cout << a[i][j] << " ";

std::cout << endl;

}

}

void print\_matrix\_to\_file(vector<vector<ll>> a, string name)

{

ofstream file(name);

for (ll i = 0; i < a.size(); i++)

{

for (ll j = 0; j < a[0].size(); j++) file << a[i][j] << " ";

file << endl;

}

file.close();

}

int main()

{

//generate\_file(200, 400, 400, 800);

string name = "200x400\_400x800";

int NUM\_THREADS;

cin >> NUM\_THREADS;

ifstream file(name + ".txt");

ll n, m;

file >> n >> m;

vector<vector<ll>> a(n, vector<ll>(m));

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

{

for (ll j = 0; j < m; j++) file >> a[i][j];

}

ll d, e;

file >> d >> e;

vector<vector<ll>> b(d, vector<ll>(e));

for (ll i = 0; i < d; i++)

{

for (ll j = 0; j < e; j++) file >> b[i][j];

}

if (NUM\_THREADS > n \* e)

{

std::cout << "Too many threads" << endl;

NUM\_THREADS = n \* e;

}

vector<vector<pair<int, int>>> indexes(NUM\_THREADS, vector<pair<int, int>>());

int count = 0;

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

{

for (int j = 0; j < e; j++)

{

indexes[count % NUM\_THREADS].push\_back({i, j});

count++;

}

}

/\*for (int i = 0; i < NUM\_THREADS; i++)

{

cout << i << "-th thread: ";

for (int j = 0; j < indexes[i].size(); j++)

{

cout << "{" << indexes[i][j].first << ", " << indexes[i][j].second << "}" << ", ";

}

cout << endl;

}\*/

vector<vector<ll>> c1 = matrix\_multiplication(a, b);

print\_matrix\_to\_file(c1, name + "\_output.txt");

vector<vector<ll>> c2 = matrix\_multiplication\_parallel(a, b, NUM\_THREADS, indexes);

print\_matrix\_to\_file(c2, name + "output\_parallel.txt");

//this\_thread::sleep\_until(chrono::high\_resolution\_clock::now() + chrono::seconds(1));

//print\_matrix(c1);

//cout << endl;

//print\_matrix(c2);

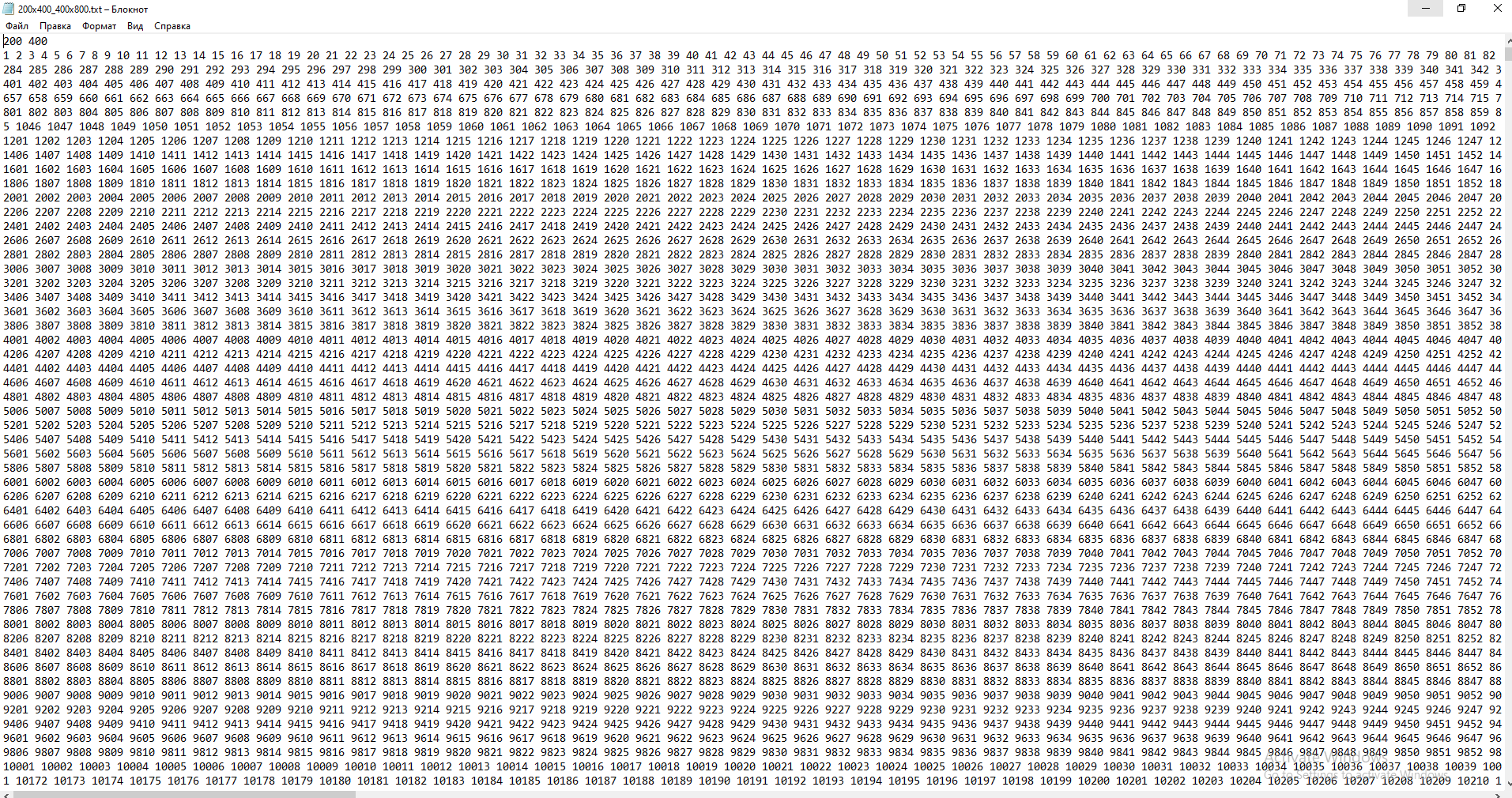
file.close();

return 0;

}

c)

200\_400\_400\_800.txt



200\_400\_400\_800\_output.txt



200\_400\_400\_800\_output\_parallel.txt



d)

|  |  |  |
| --- | --- | --- |
| К-сть потоків | Час виконання функції (в мілісекундах) | Час виконання функції (паралел.) (в мілісекундах) |
| 2 | 17929 | 14199 |
| 4 | 17994 | 7618 |
| 8 | 18065 | 6096 |
| 20 | 18464 | 13722 |
| 100 | 18102 | 13779 |
| 1000 | 18005 | 13802 |