**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

Інститут **ІКНІ**

Кафедра **ПЗ**



**ЗВІТ**

До лабораторної роботи № 7

**На тему:** *“Синхронізація потоків в ОС Windows”*

**З дисципліни:** *“Операційні системи”*

**Лектор:**

Асист. каф. ПЗ

Грицай О.Д.

**Виконав:**

ст. гр. ПЗ-21

Мартинюк Н.В.

**Прийняла:**

Асист. каф. ПЗ

Грицай О.Д.

«\_\_» \_\_\_\_\_\_\_\_\_ 2020 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів-2020

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

**Мета:** Ознайомитись зі способами синхронізації потоків. Навчитися організовувати багатопоточність з використанням синхронізації за допомогою функцій

**Завдання**

1. Реалізувати алгоритм.

2. Здійснити розпаралелювання даних алгоритмів на кілька потоків із використанням синхронізації.

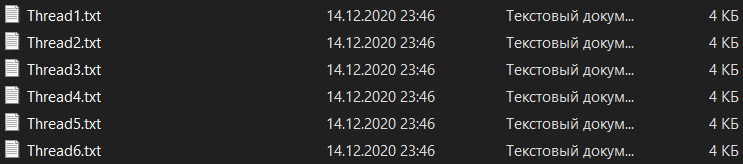
3. Реалізувати прогрес(хід) виконання задачі.

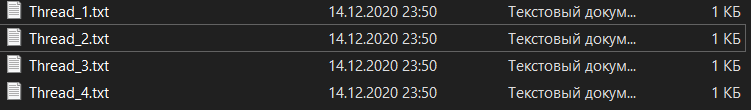
4. Для синхронізації потоків використати такі методи: Interlocked-функції, м’ютекси, семафори, критична секція, спін-блокування, Wait-функції.(на вибір для кожного завдання різні).

5. Результати виконання роботи відобразити у звіті.

**Хід роботи**

1. Розпаралелювання алгоритмів на потоки.





2. Код синхронізації за допомогою семафора або м’ютекса.

#include <iostream>

#include <Windows.h>

#include <chrono>

#include <fstream>

#include <string>

using namespace std;

void(\_stdcall\* FuncPtr)(int\* curent\_thread);

HANDLE\* handles;

HANDLE hMutex;

HANDLE hSemaphore;

int step;

int iteration = 0;

chrono::duration<double> totalTime = chrono::milliseconds::zero();

void WINAPI func(int\* currentThread)

{

//WaitForSingleObject(hMutex, INFINITE);

WaitForSingleObject(hSemaphore, INFINITE);

string name = "Thread" + to\_string(\*currentThread + 1) + ".txt";

ofstream out(name);

int start = step \* (\*currentThread);

int end = step \* ((\*currentThread) + 1);

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

for (size\_t i = start; i < end; i++)

{

out << iteration << "Martyniuk Nazar\n";

iteration++;

}

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

totalTime += endTime - startTime;

out << totalTime.count();

//ReleaseMutex(hMutex);

ReleaseSemaphore(hSemaphore, 1, nullptr);

}

int main()

{

DWORD ThreadID;

int param;

int numberOfThreads;

FuncPtr = func;

//hMutex = CreateMutexA(nullptr, FALSE, nullptr);

hSemaphore = CreateSemaphoreA(NULL, 1, 1, NULL);

cout << "Enter number of iteration: ";

cin >> param;

cout << "Enter number of threads: ";

cin >> numberOfThreads;

int\*\* parametrs = new int\*[numberOfThreads];

for (size\_t i = 0; i < numberOfThreads; i++)

{

parametrs[i] = new int;

parametrs[i][0] = i;

}

system("cls");

step = param / numberOfThreads;

handles = new void\* [numberOfThreads];

for (size\_t i = 0; i < numberOfThreads; i++)

{

if (!(handles[i] = CreateThread(0, 0, (LPTHREAD\_START\_ROUTINE)FuncPtr, parametrs[i], CREATE\_SUSPENDED, &ThreadID)))

{

cout << "Thread didn't created" << endl;

return 0;

}

ResumeThread(handles[i]);

}

WaitForMultipleObjects(numberOfThreads, handles, TRUE, INFINITE);

for (size\_t i = 0; i < numberOfThreads; i++)

{

CloseHandle(handles[i]);

}

return 0;

}

3. Синхронізація потоків за допомогою критичної секції або Interlocked-функцій.

#include <iostream>

#include <chrono>

#include <string>

#include <Windows.h>

#include <fstream>

using namespace std;

void(\_stdcall\* FuncPtr)(int\* curent\_thread);

int start = 0;

int ending = 3;

int steps;

HANDLE\* handles;

CRITICAL\_SECTION cs;

DWORD ThreadID;

volatile unsigned locked = false;

void WINAPI func(int\* currentThread)

{

string nameOfFile = to\_string(\*currentThread + 1) + ".txt";

ofstream out("Thread\_" + nameOfFile);

int n = steps;

double eps = 0.1;

double x = start;

double step = (double)(start + ending) / n;

while (InterlockedExchange(&locked, 1) == 1)

Sleep(0);

//EnterCriticalSection(&cs);

while (x < ending)

{

double formula = cos(x);

double result = 0;

double diff = 0;

int n = 1;

double tmp;

do

{

int fact = 1, mult = 2 \* n;

while (mult != 1) {

fact \*= mult;

mult--;

}

tmp = diff;

diff = (pow(-1, n) \* (pow(x, (2 \* n)) / (fact)));

result += diff;

n++;

} while (fabs(diff - tmp) > eps);

result += 1;

x += step;

out << "X = " << x << ", formula = " << formula << ", result = " << result << "\n";

}

//LeaveCriticalSection(&cs);

InterlockedExchange(&locked, 0);

}

int main()

{

int numberOfThreads;

cout << "Enter count of step: ";

cin >> steps;

cout << "Enter count of threads: ";

cin >> numberOfThreads;

handles = new void\* [numberOfThreads];

int\*\* param = new int\* [numberOfThreads];

for (size\_t i = 0; i < numberOfThreads; i++)

{

param[i] = new int;

param[i][0] = i;

}

FuncPtr = func;

InitializeCriticalSection(&cs);

int step = steps / numberOfThreads;

steps = step;

for (size\_t i = 0; i < numberOfThreads; i++)

{

if (!(handles[i] = CreateThread(0, 0, (LPTHREAD\_START\_ROUTINE)FuncPtr, param[i], 0, &ThreadID)))

cout << "Thread did'nt created\n";

steps += step;

}

for (size\_t i = 0; i < numberOfThreads; i++)

CloseHandle(handles[i]);

return 0;

}

**Висновок**

Виконуючи дану лабораторну роботу, я навчивсь синхронізувати створені потоки за допомогою м’ютекса, семафору, критичної секції та Interlocked-функцій.