МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

Кафедра комп’ютерних наук та інформаційних систем

**Лабораторна робота №3**

з курсу «Паралельні та розподілені обчислення»

на тему:

«Синхронізація паралельних операцій за допомогою механізму подій»

**Виконав:**

студент групи КН-41

Сенів П.В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 р.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

**Перевірив:**

викладач кафедри КНІС

к.т.н., Ізмайлов А. В. «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 р.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оцінка, підпис)

м. Івано-Франківськ

2021

***Мета роботи*:** Навчитись синхронізувати паралельні операції за допомогою  механізму подій.

**Теоретичні відомості**

Одним із способів синхронізації паралельних операцій є проста перевірка  спільного для потоків прапорця, який захищений м'ютексом, вважаючи, що один  із потоків підніме цей прапорець, коли завершить свою операцію. Такий підхід  має ряд недоліків, оскільки, передбачає витрачання ресурсів системи на постійну  перевірку значення прапорця. Одним із способів подолання цього недоліку є  застосування функції this\_thread::sleep\_for(), яка змушує потік «спати» між  перевірками значення прапорця. Наприклад, за допомогою наступної  конструкції:

unique\_lock<mutex> lk(m);

while (!flag)

{

lk.unlock();

this\_thread::sleep\_for(chrono::milliseconds(100));

lk.lock();

}

Наведений спосіб не позбавлений більшості недоліків, пов’язаних з  опитуванням значень прапорів, тому рекомендованим способом синхронізації  потоків є використання засобів стандартної бібліотеки С++, які дозволяють  потоку очікувати на події. Найпростіший механізм очікування на події, які  виникають у іншому потоці (наприклад, поява нового завдання), дають умовні  змінні. Концептуально, умовна змінна асоційована з якоюсь подією або іншою умовою, причому, один або кілька потоків можуть чекати, коли ця умова почне  виконуватись. Якщо деякий потік вирішить, що умова виконана, він може  сповістити про це один або кілька потоків, які очікують умовну змінну, в  результаті чого, вони відновлять роботу.

Стандартна бібліотека С++ надає не одну, а дві реалізації умовних змінних:  std::condition\_variable і std::condition\_variable\_any. Обидва класи оголошені в  заголовку <condition\_variable>. В обох випадках для забезпечення синхронізації  необхідна взаємодія з м’ютексами; перший клас може працювати тільки з  std::mutex, другий – з будь-яким класом, який відповідає мінімальним вимогам  до «м’ютексоподібності». Оскільки, клас std::condition\_variable\_any є більш  загальним, то його використання може обійтися дорожче з точки зору обсягу  споживаної пам'яті, продуктивності та ресурсів операційної системи. Тому, якщо додаткова гнучкість не потрібна, то краще обмежитися класом  std::condition\_variable.

**Хід роботи:**

1.

**Код:**

//Task1

#include <iostream>

#include <thread>

#include <mutex>

#include <queue>

#include <Windows.h>

using namespace std;

bool isEnd = false;

mutex mut;

queue<int> q;

int primeNumber(unsigned long a, unsigned long i)

{

return i \* i <= a ? (a % i) && primeNumber(a, i + 1) : 1;

}

void dataPreparation() {

int count = 0;

cout << "Enter the number of numbers: ";

cin >> count;

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

{

int value = 0;

cout << "Enter number: ";

cin >> value;

q.push(value);

}

isEnd = true;

}

void dataProcessing() {

unique\_lock<std::mutex> lk(mut);

while (!isEnd) {

lk.unlock();

this\_thread::sleep\_for(chrono::milliseconds(100));

lk.lock();

}

cout << "Processing" << endl;

while (!q.empty()) {

int v = q.front();

q.pop();

auto isPrime = primeNumber((long)v, (long)2);

cout << "Number: " << v << " is prime = " << (isPrime == 1 ? "True" : "False") << endl;

}

}

int main()

{

thread DataPreparation(dataPreparation);

thread DataProcessing(dataProcessing);

DataPreparation.detach();

DataProcessing.join();

}

**Результати виконання роботи**:



2)

**Код:**

//Task2

#include <iostream>

#include <thread>

#include <mutex>

#include <queue>

#include <Windows.h>

#include <condition\_variable>

using namespace std;

condition\_variable dataCondition;

mutex mut;

int i = 0;

void Awake() {

this\_thread::sleep\_for(chrono::milliseconds(100));

cout << "Informing" << endl;

dataCondition.notify\_all();

this\_thread::sleep\_for(chrono::milliseconds(100));

i = 1;

cout << "Updated i=1. Informing" << endl;

dataCondition.notify\_all();

}

void Waits() {

unique\_lock<mutex> lk(mut);

cout << "Go to standby" << endl;

dataCondition.wait(lk, [] {return i == 1; });

cout << "The waiting state is over" << endl;

}

int main() {

thread waits1(Waits);

thread waits2(Waits);

thread waits3(Waits);

thread awakeness(Awake);

waits1.detach();

waits2.detach();

waits3.detach();

awakeness.join();

Sleep(1000);

}

**Результати виконання роботи**:



Після виконання notify\_all без зміни значення, яке є глобальною, жоден потік не виконується. Після зміни значення змінної та виклику notify\_all. Виконуються всі потоки.

3)

**Код:**

//Task3

#include <iostream>

#include <thread>

#include <mutex>

#include <queue>

#include <Windows.h>

#include <condition\_variable>

using namespace std;

condition\_variable dataCondition;

mutex mut;

int i = 0;

void Notify() {

this\_thread::sleep\_for(chrono::milliseconds(200));

cout << "Informing" << endl;

dataCondition.notify\_one();

i = 1;

cout << "Updated i=1. Informing" << endl;

dataCondition.notify\_one();

}

void Waits(int n) {

unique\_lock<mutex> lk(mut);

cout << "Go to standby from the stream: " << n << endl;

dataCondition.wait(lk, [] {return i == 1; });

cout << "Message from the thread: " << n << endl;

}

int main() {

thread waits1(Waits, 1);

thread waits2(Waits, 2);

thread waits3(Waits, 3);

thread awakeness(Notify);

waits1.detach();

waits2.detach();

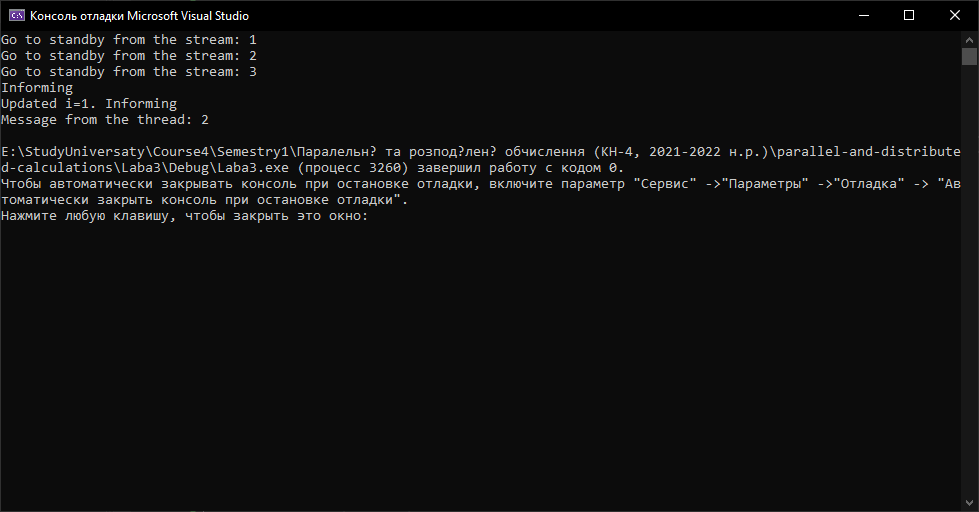
waits3.detach();

awakeness.join();

Sleep(1000);

}

**Результати виконання роботи:**



Якщо викликати notify\_one перед заданням, то нічого не відбувається. Якщо викликати після задання, то виконується 1 потік з всіх.

4)

**Код:**

//Task4

#include <iostream>

#include <thread>

#include <mutex>

#include <queue>

#include <Windows.h>

using namespace std;

bool isEnd = false;

mutex mut;

condition\_variable dataCondition;

queue<int> q;

int primeNumber(unsigned long a, unsigned long i)

{

return i \* i <= a ? (a % i) && primeNumber(a, i + 1) : 1;

}

void dataPreparation() {

cout << "False informing" << endl;

dataCondition.notify\_all();

int count = 0;

cout << "Enter the number of numbers: ";

cin >> count;

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

{

int value = 0;

cout << "Enter number: ";

cin >> value;

q.push(value);

}

cout << "Correct notification" << endl;

dataCondition.notify\_all();

}

void dataProcessing() {

unique\_lock<std::mutex> lk(mut);

cout << "I started waiting" << endl;

dataCondition.wait(lk, [] {return !q.empty(); });

cout << "Processing started" << endl;

while (!q.empty()) {

int v = q.front();

q.pop();

auto isPrime = primeNumber((long)v, (long)2);

cout << "Number: " << v << " is prime = " << (isPrime == 1 ? "True" : "False") << endl;

}

}

int main()

{

thread DataPreparation(dataPreparation);

thread DataProcessing(dataProcessing);

DataPreparation.detach();

DataProcessing.join();

}

**Результати виконання роботи**:



**Висновок:** на лабораторній роботі я навчився синхронізувати паралельні операції за допомогою  механізму подій. Було реалізовано два підходи до синхронізації паралельних операції. Підхід з використанням умовних змінних являється більш зручним порівняно з прапорцем, оскільки дозволяє передати в функцію очікування лямбда-функцію для перевірки умови. Мої вихідні файли з проектом знаходяться за адресою: <https://github.com/PavloSeniv/parallel-and-distributed-calculations/tree/main/Laba3>