МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

Кафедра комп’ютерних наук та інформаційних систем

**Лабораторна робота №4**

з курсу «Паралельні та розподілені обчислення»

на тему:

«Синхронізація паралельних операцій за допомогою механізму майбутніх результатів»

**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 майбутніх результатів.

**Теоретичні відомості**

У класі std::thread не передбачено механізму передачі в основний потік (або  інше місце виклику) результатів виконання функції-потоку. Для цього  застосовується механізм майбутніх результатів. У стандартній бібліотеці С++ є  два різновиди майбутніх результатів, які реалізовані у формі двох шаблонів  класів, які оголошені в заголовку <future>: унікальні майбутні результати  (std::future<>) і розділювані майбутні результати (std::shared\_future<>).

Одним із засобів повернення результатів з потоку є шаблон функції  std::async (оголошений у заголовку <future>). Функція std::async дозволяє  запустити асинхронне завдання, у результаті якого у поточний момент немає  необхідності. Замість об’єкта std::thread вона повертає об’єкт std::future, який  буде містити повернене значення, коли воно стане доступним. Коли програмі  знадобиться значення, вона викликає функцію-член get() об’єкта-майбутнього, і  тоді потік буде призупинено до готовності майбутнього результату, після чого  він поверне значення.

Наприклад, для деякої функції з двома аргументами запуск std::async і  отримання результатів матиме вигляд:

future<int> f1 = async(launch::async, Func, arg1, arg2);

int res = f1.get();

Поведінку std::async можна задати у додатковому параметрі перед  функцією, яка викликається. Цей параметр має тип std::launch і може набувати  таких значень: std::launch::deferred – відкласти виклик функції до того моменту,  коли буде викликана функція-член wait() або get() об’єкта-майбутнього;

std::launch::async – запускати функцію в окремому потоці.

**Хід роботи:**

1.

**Код:**

//Task1

#include <iostream>

#include <future>

#include <chrono>

using namespace std;

int getPrime(int n);

int root\_number(int n);

int main()

{

int number = 1;

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

cin >> number;

std::chrono::steady\_clock::time\_point begin = std::chrono::steady\_clock::now();

future<int> result = async(launch::async, getPrime, number);

cout << "Calculate root of " << number <<". Next enter 1" << endl;

int choice = 0;

cin >> choice;

if (choice == 1)

{

cout << "Square of " << number << ": " << root\_number(number) << endl;

}

int simple = result.get();

cout << "Prime number is: " << simple << endl;

std::chrono::steady\_clock::time\_point end = std::chrono::steady\_clock::now();

std::cout << "The time difference = " << std::chrono::duration\_cast<std::chrono::milliseconds>(end - begin).count() << "[ms]" << std::endl;

std::cout << "The time difference = " << std::chrono::duration\_cast<std::chrono::seconds> (end - begin).count() << "[s]" << std::endl;

system("pause");

}

int root\_number(int n) {

return n \* n;

}

int getPrime(int n) {

int count = 0;

int max\_limit = 100;

if (max\_limit < n)

{

max\_limit = n;

}

int i = 1;

while (count != n)

{

int kol = 0;

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

if (i % j == 0) kol++;

if (kol == 1 || kol == 2) {

count++;

}

if (count == n)

{

return i;

}

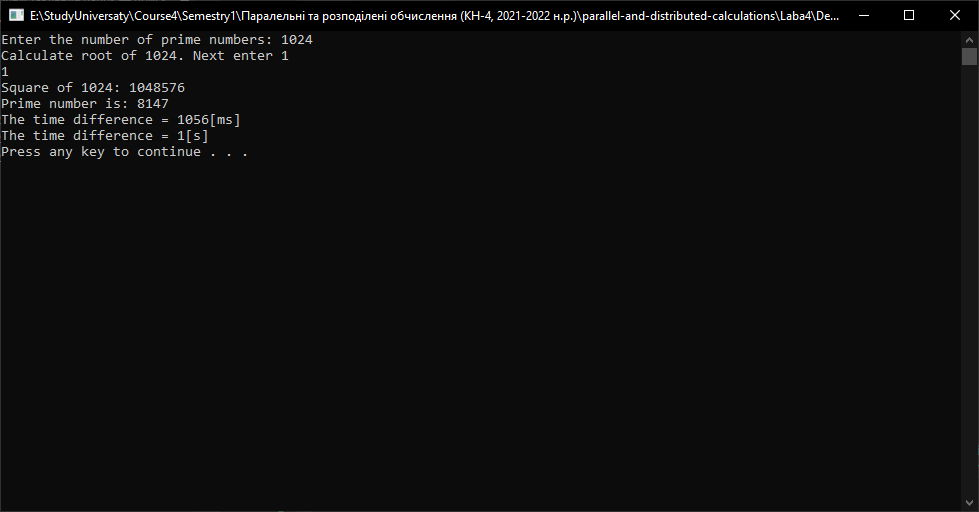
i++;

}

return -1;

}

**Результати виконання роботи**:



Async являється оптимальнішим, оскільки обрахування починається до того як користувач введе дані.

2)

**Код:**

//Task2

#include <iostream>

#include <future>

#include <chrono>

#include <deque>

#include <queue>

using namespace std;

int getPrime(int n);

int root\_number(int n);

void task1();

void task2();

bool stop = false;

void dataProcessing();

queue<packaged\_task<int(int)>> qTask;

queue<int> qNumber;

int main()

{

task2();

system("pause");

}

void task2() {

thread processing(dataProcessing);

while (!stop)

{

int number = 1;

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

cin >> number;

if (number == -1)

stop = true;

packaged\_task<int(int)> task(getPrime);

qTask.push(move(task));

qNumber.push(number);

}

processing.join();

}

void dataProcessing() {

while (!stop)

{

if (!qTask.empty())

{

packaged\_task<int(int)> task = move(qTask.front());

qTask.pop();

int N = qNumber.front();

qNumber.pop();

task(N);

}

}

}

int getPrime(int n) {

int count = 0;

int max\_limit = 100;

if (max\_limit < n)

{

max\_limit = n;

}

int i = 1;

while (count != n)

{

int kol = 0;

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

if (i % j == 0) kol++;

if (kol == 1 || kol == 2) {

count++;

}

if (count == n)

{

cout << "---Current number: " << n << " prime is " << i << endl;

return i;

}

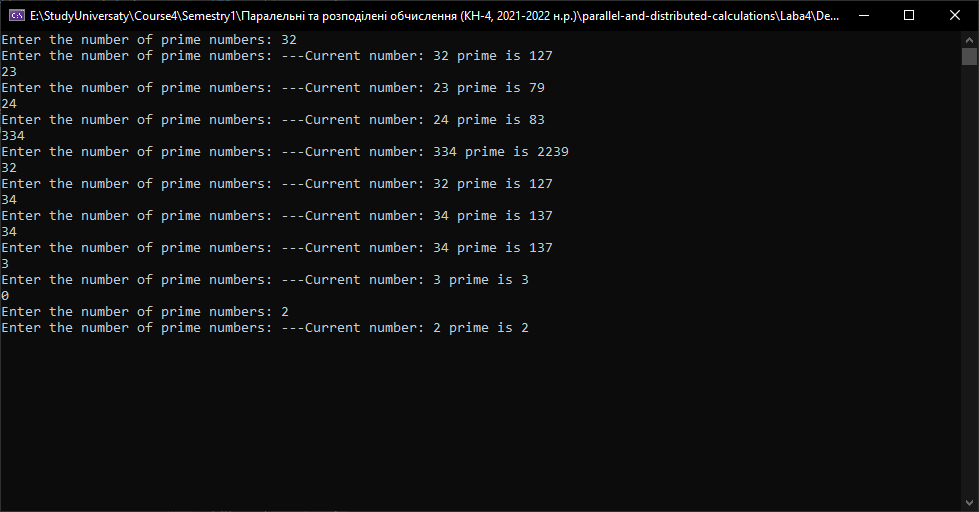
i++;

}

return -1;

}

**Результати виконання роботи**:



3)

**Код:**

#include <iostream>

#include <future>

#include <chrono>

#include <deque>

#include <queue>

using namespace std;

int getPrime(int n);

void task3();

void currentOne(int N, promise<int> primePromise, promise<int> primePromise10, promise<bool> isReadyPromise) {

future<int> result = async(launch::async, getPrime, N);

int res = result.get();

move(primePromise).set\_value(res);

move(isReadyPromise).set\_value(true);

move(primePromise10).set\_value(res \* 10);

}

void currentTwo(future<bool> isReadyPromise, shared\_future<int> primePromise) {

auto res = isReadyPromise.get();

if (res)

{

this\_thread::sleep\_for(2000ms);

auto n = primePromise.get();

cout << "Mul is " << n \* n << endl;

}

}

int main()

{

task3();

system("pause");

}

void task3() {

promise<int> ntePrime;

promise<int> nPrimex10;

promise<bool> isReady;

shared\_future<int> primeFuture = ntePrime.get\_future();

auto nPrime10Future = nPrimex10.get\_future();

auto isReadyFuture = isReady.get\_future();

int Nte = 1;

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

cin >> Nte;

thread t1(currentOne, Nte, move(ntePrime), move(nPrimex10), move(isReady));

thread t2(currentTwo, move(isReadyFuture), primeFuture);

t1.detach();

t2.detach();

int nValue = primeFuture.get();

cout << "Prime number is " << nValue << endl;

int n10Value = nPrime10Future.get();

cout << "Number mul 10 is " << n10Value << endl;

system("pause");

}

int getPrime(int n) {

int count = 0;

int limit = 100;

if (limit < n)

{

limit = n;

}

int i = 1;

while (count != n)

{

int kol = 0;

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

if (i % j == 0) kol++;

if (kol == 1 || kol == 2) {

count++;

}

if (count == n)

{

return i;

}

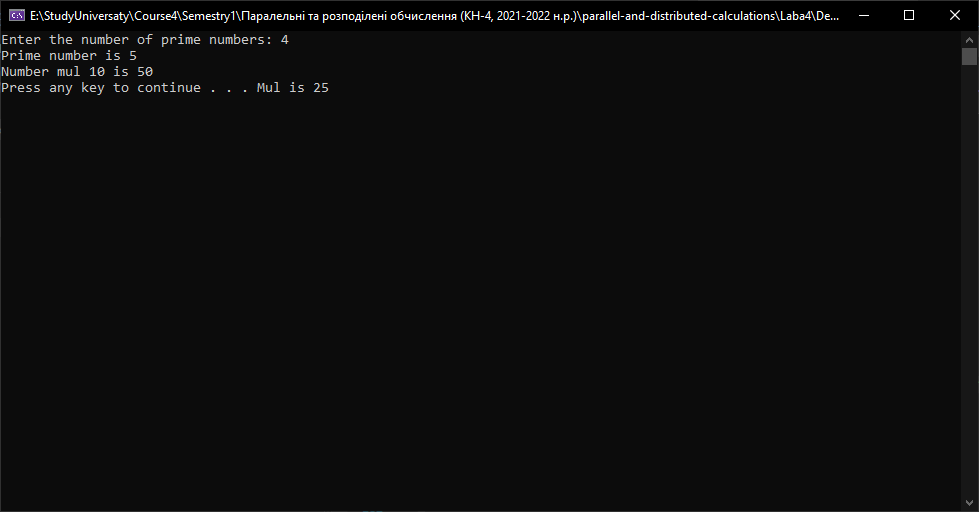
i++;

}

return -1;

}

**Результати виконання роботи:**



**Висновок:** на лабораторній роботі я навчився синхронізувати паралельні операції за допомогою  механізму майбутніх результатів. Досліджено механізми Future, Promise, async, packaged\_task. Мої вихідні файли з проектом знаходяться за адресою: <https://github.com/PavloSeniv/parallel-and-distributed-calculations/tree/main/Laba4>