МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

Кафедра комп’ютерних наук та інформаційних систем

**Лабораторна робота №1**

з курсу «Паралельні та розподілені обчислення»

на тему:

«Застосування м’ютексів у паралельних обчисленнях»

**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 застосовувати м’ютекси для забезпечення ізолювання  доступу потоків до розділюваних даних.

**Теоретичні відомості**

Паралелізм – це, у загальному випадку, одночасне виконання системою  декількох завдань. У операційній системі паралельне виконання завдань  забезпечується застосуванням процесів і потоків, які виконуються паралельно.  Для роботи з потоками у мові високого рівня С++ застосовуються класи, які  розміщені у заголовку thread.h.

При паралельній обробці даних виникає ряд проблем синхронізації  доступу різних потоків до одних і тих самих даних. Одним з механізмів  забезпечення розділеного у часі доступу потоків до даних є м’ютекс. М’ютекс – примітив синхронізації, який забезпечує синхронізацію потоків шляхом  забезпечення доступу до заданого ресурсу у заданий час лише одного потоку та  блокування доступу до нього інших потоків. Перед тим, як звернутись до даних,  програма (потік) захоплює м’ютекс, а після виконання всіх операцій, звільняє  його. Бібліотека Thread Library гарантує, що якщо один потік захопив деякий м’ютекс, то решта потоків, які намагаються захопити цей м’ютекс, повинні  будуть очікувати його звільнення. Результатом є інваріантний доступ до даних.

**Хід роботи:**

1.

**Код:**

#include <iostream>

#include <thread>

#include <mutex>

#include <Windows.h>

using namespace std;

void thread\_for1() {

cout << 1 << endl;

}

void thread\_for2() {

cout << 2 << endl;

}

int main()

{

thread t1(thread\_for1);

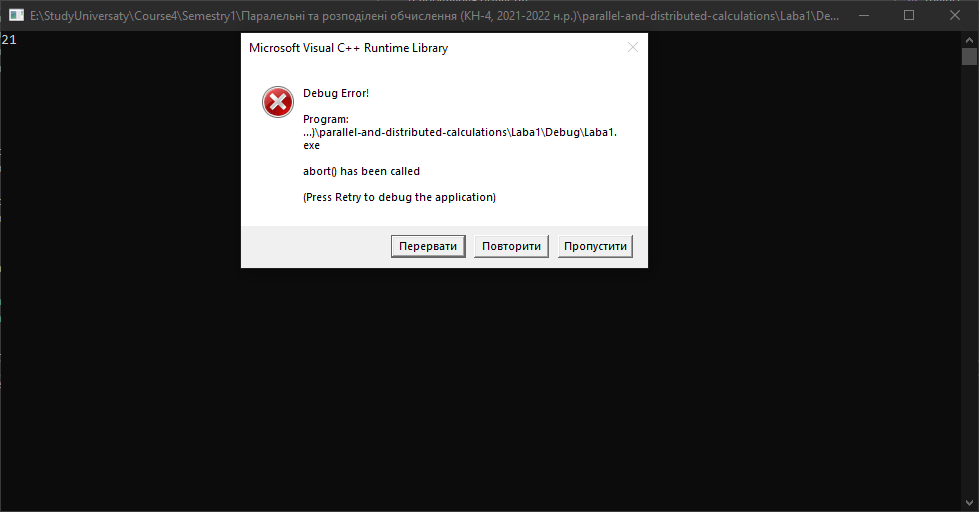
thread t2(thread\_for2);

Sleep(1000);

return 0;

}

**Результати виконання роботи**:



Програма після дебагу видає помилку.

2)

**Код:**

#include <iostream>

#include <thread>

#include <mutex>

#include <Windows.h>

using namespace std;

void thread\_for1() {

cout << 1 << endl;

}

void thread\_for2() {

cout << 2 << endl;

}

int main()

{

thread t1(thread\_for1);

thread t2(thread\_for2);

Sleep(1000);

t1.detach();

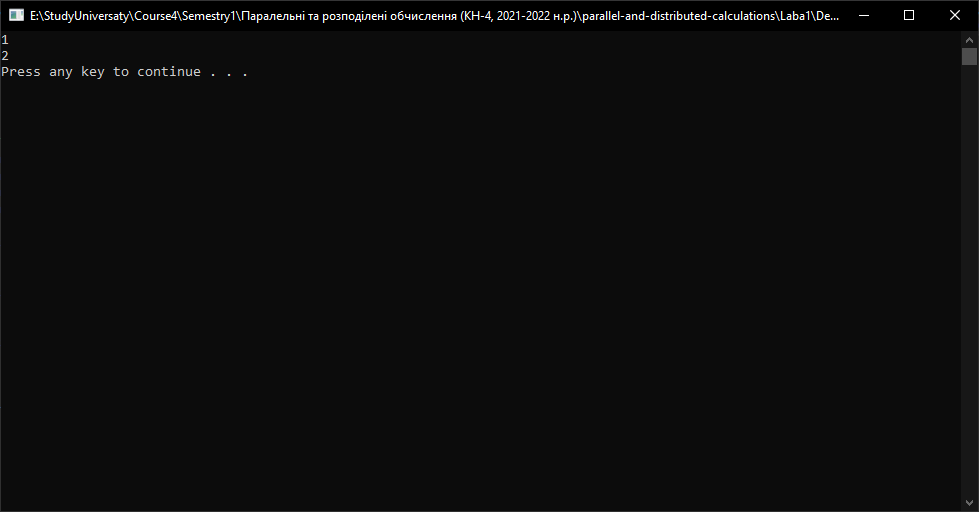
t2.detach();

system("pause");

return 0;

}

**Результати виконання роботи**:



Після кожного дебагу програми результати міняються з 1 2 на 2 1

3)

**Код:**

#include <iostream>

#include <thread>

#include <mutex>

#include <list>

#include <algorithm>

using namespace std;

list<int> ls;

void AddToList(int number)

{

for (int i = number; i < 10 + number; i++)

{

ls.push\_back(i);

cout << "Added: " << i << endl;

}

}

void ListContains(int number)

{

for (int i = 0; i < 10; i++)

{

bool found = (find(ls.begin(), ls.end(), number) != ls.end());

if (found)

{

cout << "True " << endl;

}

else {

cout << "False " << endl;

}

}

}

int main() {

thread t1(AddToList, 12);

thread t2(ListContains, 18);

t1.join();

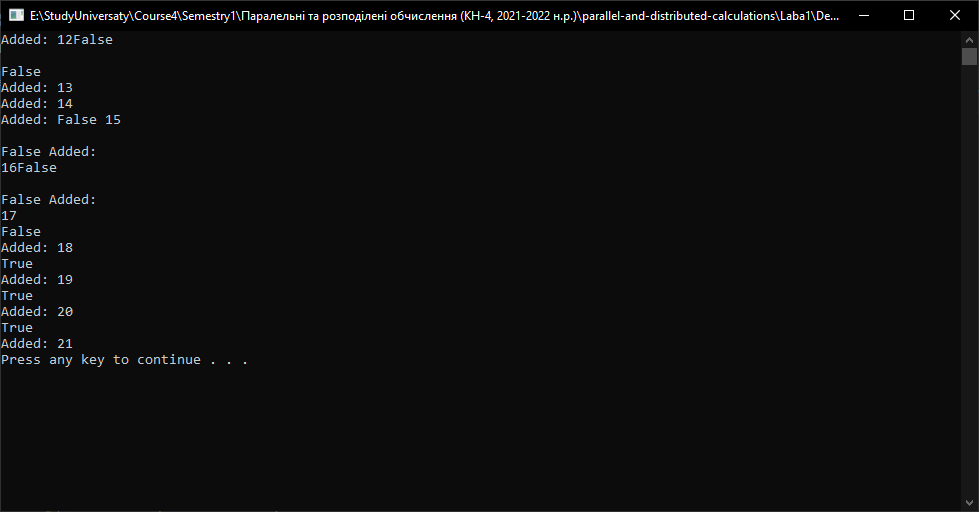
t2.join();

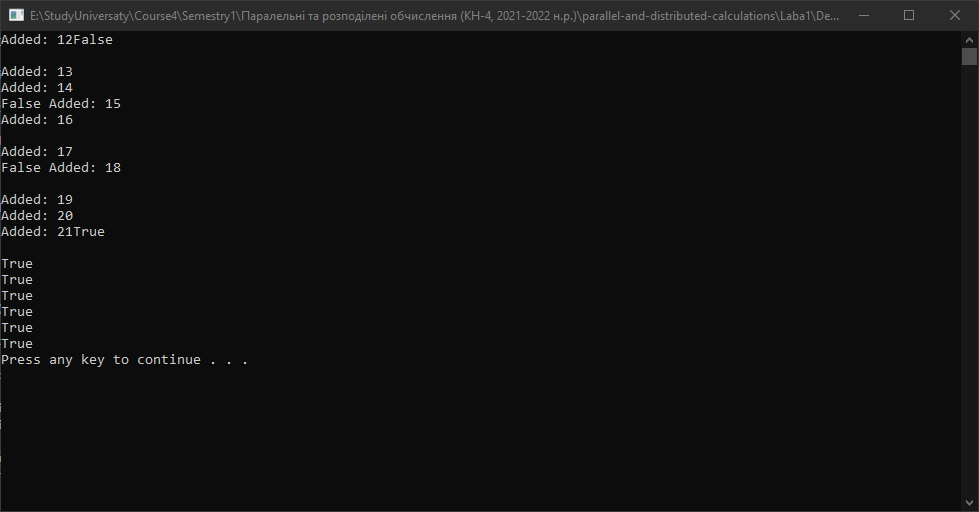
system("pause");

return 0;

}

**Результати виконання роботи:**





Програма кожного разу виводить дані по різному. Причиною є два потоки, які працюють одночасно. Вони одне одному заважають виводити повідомлення і відбувається те, що ми бачимо на скріншотах.

4)

**Код:**

#include <iostream>

#include <thread>

#include <mutex>

#include <list>

#include <algorithm>

using namespace std;

mutex m1;

list<int> ls;

void AddToList(int number)

{

for (int i = number; i < 10 + number; i++)

{

m1.lock();

ls.push\_back(i);

cout << "Added: " << i << endl;

m1.unlock();

}

}

void ListContains(int number)

{

for (int i = 0; i < 10; i++)

{

m1.lock();

bool found = (find(ls.begin(), ls.end(), number) != ls.end());

if (found)

{

cout << "True " << endl;

}

else {

cout << "False " << endl;

}

m1.unlock();

}

}

int main() {

thread t1(AddToList, 12);

thread t2(ListContains, 18);

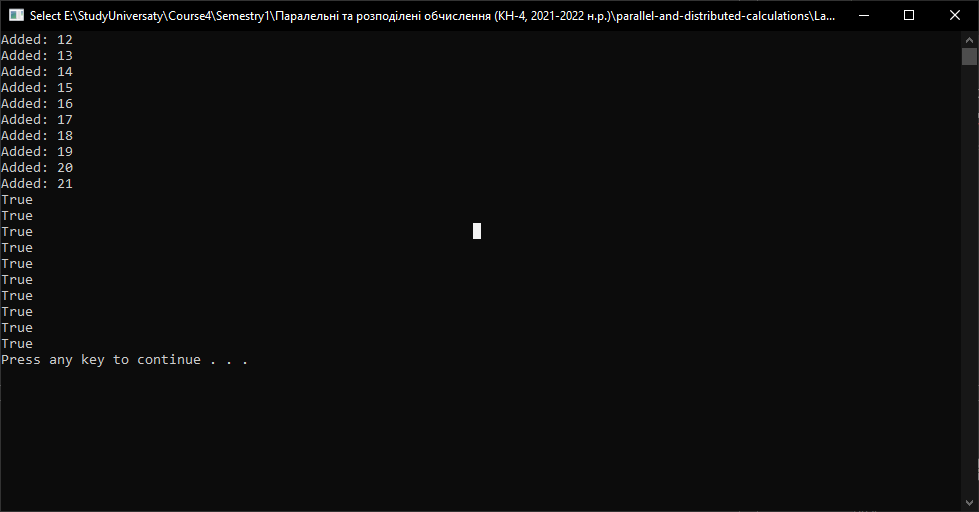
t1.join();

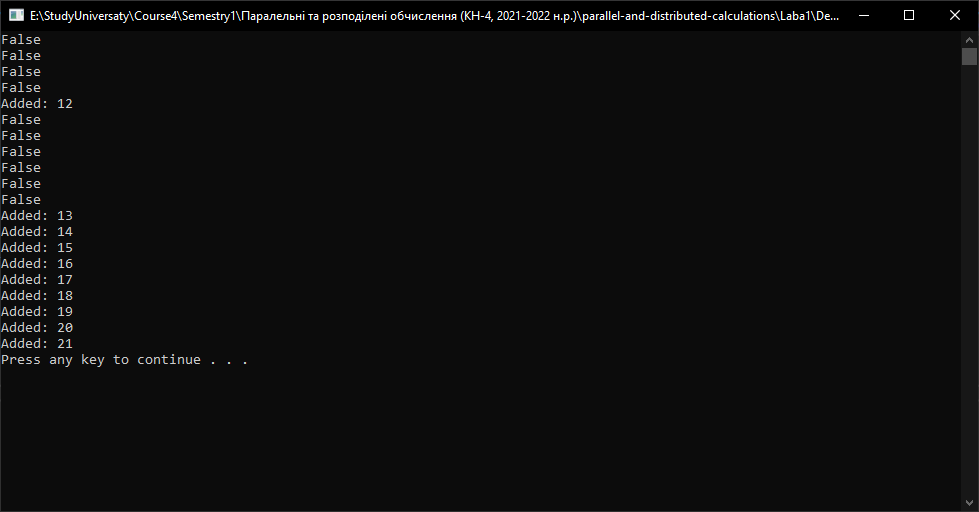
t2.join();

return 0;

}

**Результати виконання роботи**:





Додавання Mutex до коду дозволяє вирішити проблему виводу повідомлення двома потоками, які заважали одне одному. Mutex забезпечив інваріантний доступ до потоків і результат цього видно на скріншотах.

**Висновок:** на лабораторній роботі я навчився застосовувати м’ютекси для забезпечення ізолювання  доступу потоків до розділюваних даних. Мої вихідні файли з проектом знаходяться за адресою: <https://github.com/PavloSeniv/parallel-and-distributed-calculations/tree/main/Laba1>