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(-ла):**

студент(-ка) групи ШИФР-1

Халупняк І. В.

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

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

(підпис)

**Перевірив:**

викладач кафедри КНІС

к.т.н. Ізмайлов А. В.

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

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

(оцінка, підпис)

м. Івано-Франківськ

2021

Застосування м’ютексів у паралельних обчисленнях

**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.

#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();

return 0;

}

1.2.1 Програма після виводу видає помилку.

1.2.2 Під час кожного запуску результати діалогового вікна міняються.

1.2.3

#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();

return 0;

}Изображение выглядит как текст

Автоматически созданное описание

Програма кожен раз працює по іншому. Через те що два потоки працюють одночасно, один потім заважає виводити повідомлення іншому.

1.2.4

#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;

}

Изображение выглядит как текст, снимок экрана, электроника, компьютер

Автоматически созданное описание

Mutex дозволяє вирішити проблему з минулого завдання.

**Висновок:** на лабораторній роботі я навчився застосовувати м’ютекси для забезпечення ізолювання  доступу потоків до розділюваних даних.