МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ   
ІМЕНІ ВАСИЛЯ СТЕФАНИКА

Кафедра комп’ютерних наук та інформаційних систем

**Лабораторна робота №2**

з курсу «Паралельні та розподілені обчислення»

на тему:

«Застосування шаблонів стандартної бібліотеки на основі м’ютексів для ізолювання доступу до даних»

**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 шаблони стандартної бібліотеки на  основі м’ютексів для ізолювання доступу потоків до розділюваних даних.

**Теоретичні відомості**

У С ++ для створення м’ютекса слід сконструювати об'єкт класу std::mutex,  для захоплення м’ютекса служить функція-член lock(), а для звільнення – функція-член unlock(). Однак, викликати ці функції безпосередньо не  рекомендується. Замість цього в стандартній бібліотеці є шаблон класу  std::lock\_guard, який захоплює м’ютекс у конструкторі і звільняє в деструкторі.  Цим гарантується, що захоплений м’ютекс обов’язково буде звільнений. Обидва  класи визначені у заголовку <mutex>.

Крім самого м’ютекса, конструктору std::lock\_guard можна передати  параметр std::adopt\_lock, який повідомляє об’єктові std::lock\_guard, що м’ютекс  уже захоплений, і йому потрібно лише прийняти володіння існуючим блокуванням, а не намагатися ще раз захопити м’ютекс у конструкторі.

У більшості випадків м’ютекс і захищені ним дані поміщають в один клас,  а не у глобальні змінні. Це є стандартним застосуванням правил об’єктно орієнтованого проектування. Якщо помістити обидві сутності у клас, то чітко  вказується їх взаємопов’язаність, а крім того, забезпечується інкапсулювання  функціональності і обмеження доступу. Тому м’ютекс і захищені ним дані  оголошують закритими полями деякого класу.

Застосування більше, ніж одного м’ютекса в одній ділянці коду породжує  не лише контроль над даними, але й ряд проблем. Нехай два потоки для  виконання деякої операції повинні захопити два м’ютекса, але склалося так, що  кожен потік захопив тільки один м’ютекс і чекає іншого. Жоден потік не може  продовжити роботу, тому що кожен чекає, поки інший звільнить потрібний йому  м’ютекс. Така ситуація називається взаємним блокуванням. Це найнебезпечніша

з проблем, які виникають, коли для виконання операції потрібно захопити більше  одного м’ютекса.

Одним із засобів захисту від взаємного блокування є функція std::lock стандартної бібліотеки, яка вміє захоплювати відразу два і більше м’ютексів без  ризику отримати взаємне блокування. При цьому слід пам’ятати, що std::lock  допомагає уникнути взаємного блокування у випадках, коли потрібно захопити  відразу два або більше м’ютексів, проте, вона не в силах допомогти, якщо  м’ютекси захоплюються порізно. У такому випадку залишається покладатися  тільки на дисципліну програмування

**Хід роботи:**

1.

**Код:**

//Task1 mutex

#include <iostream>

#include <thread>

#include <mutex>

#include <Windows.h>

#include <list>

#include <algorithm>

using namespace std;

list<int> l;

mutex m1;

void AddToList(int number){

m1.lock();

l.push\_back(number);

cout << number << " - pushed!\n";

m1.unlock();

}

void ListContains(int number){

m1.lock();

auto exist = find(l.begin(), l.end(), number);

if (exist != l.end()) {

cout << number << " - exist\n";

}

else

{

cout << number << " - not exist\n";

}

m1.unlock();

}

int main() {

auto number = 16;

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

{

thread t1(AddToList, number);

thread t2(ListContains, 19);

number++;

t1.join();

t2.join();

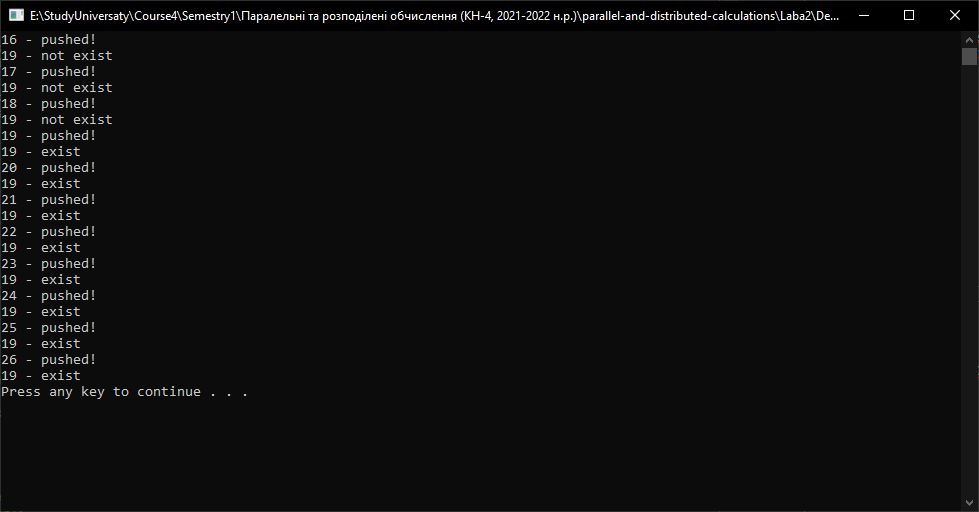
}

system("pause");

return 0;

}

**Результати виконання роботи**:



**Код:**

//Task1 lock\_guard

#include <iostream>

#include <thread>

#include <mutex>

#include <Windows.h>

#include <list>

#include <algorithm>

using namespace std;

list<int> l;

mutex m1;

void AddToList(int number){

lock\_guard<std::mutex> guard(m1);

l.push\_back(number);

cout << number << " - pushed!\n";

}

void ListContains(int number){

lock\_guard<std::mutex> guard(m1);

auto exist = find(l.begin(), l.end(), number);

if (exist != l.end()) {

cout << number << " - exist\n";

}

else

{

cout << number << " - not exist\n";

}

}

int main() {

auto number = 16;

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

{

thread t1(AddToList, number);

thread t2(ListContains, 19);

number++;

t1.join();

t2.join();

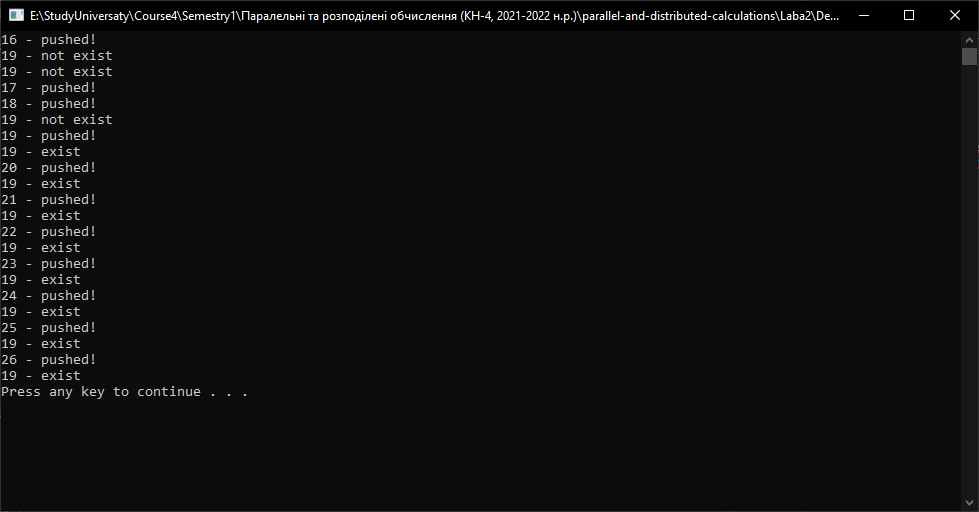
}

system("pause");

return 0;

}

**Результати виконання роботи**:



При запуску декілька разів програм я отримував правильні результати і програми виконували те, що потрібно. Проте, якщо між викликами функцій lock() та unlock() буде непередбачувана ситуація, то функція unlock() просто не виконається. Через це краще застосовувати lock\_guard, який забезпечить звільнення мютекса будь-коли.

2)

**Код:**

#include <iostream>

#include <thread>

#include <mutex>

#include <Windows.h>

#include <list>

#include <algorithm>

using namespace std;

class someData {

public:

string firstName;

string lastName;

string address;

int age;

};

class exchangePerson {

public:

someData data;

mutex myMutex;

static void JohnDoe(exchangePerson& person) {

lock\_guard<mutex> guard(person.myMutex);

person.data.firstName = "John";

person.data.lastName = "Doe";

person.data.address = "Unknown";

person.data.age = 120;

}

static void JacobSmith(exchangePerson& person) {

lock\_guard<mutex> guard(person.myMutex);

person.data.firstName = "Jacob";

person.data.lastName = "Smith";

person.data.address = "Known";

person.data.age = 1;

}

static void Swap(exchangePerson& p1, exchangePerson& p2) {

Show(p1, "Person 1 before swap");

Show(p2, "Person 2 before swap");

if (!AreEqual(p1, p2)){

lock(p1.myMutex, p2.myMutex);

lock\_guard<mutex> lock1(p1.myMutex, adopt\_lock);

lock\_guard<mutex> lock2(p2.myMutex, adopt\_lock);

exchangePerson newPerson;

newPerson.data.firstName = p1.data.firstName;

newPerson.data.lastName = p1.data.lastName;

newPerson.data.address = p1.data.address;

newPerson.data.age = p1.data.age;

p1.data.firstName = p2.data.firstName;

p1.data.lastName = p2.data.lastName;

p1.data.address = p2.data.address;

p1.data.age = p2.data.age;

p2.data.firstName = newPerson.data.firstName;

p2.data.lastName = newPerson.data.lastName;

p2.data.address = newPerson.data.address;

p2.data.age = newPerson.data.age;

Show(p1, "Person 1 after swap");

Show(p2, "Person 2 after swap");

}

}

static void Show(exchangePerson& person, string text) {

cout << text << " -> firstName - " << person.data.firstName << ", lastName - " << person.data.lastName << ", age - " << person.data.age << ", address - " << person.data.address << "\n";

}

static bool AreEqual(const exchangePerson& a, const exchangePerson& b){

return addressof(a) == addressof(b);

}

};

void JohnDoe(exchangePerson& person){

person.JohnDoe(person);

person.Show(person, "Person 1");

}

void JacobSmith(exchangePerson& person){

person.JacobSmith(person);

person.Show(person, "Person 2");

}

void Swap(exchangePerson& person1, exchangePerson& person2){

person1.Swap(person1, person2);

person2.Swap(person1, person2);

}

int main() {

exchangePerson person1;

exchangePerson person2;

thread t1(JohnDoe, ref(person1));

t1.join();

thread t2(JacobSmith, ref(person2));

t2.join();

thread t3(Swap, ref(person1), ref(person2));

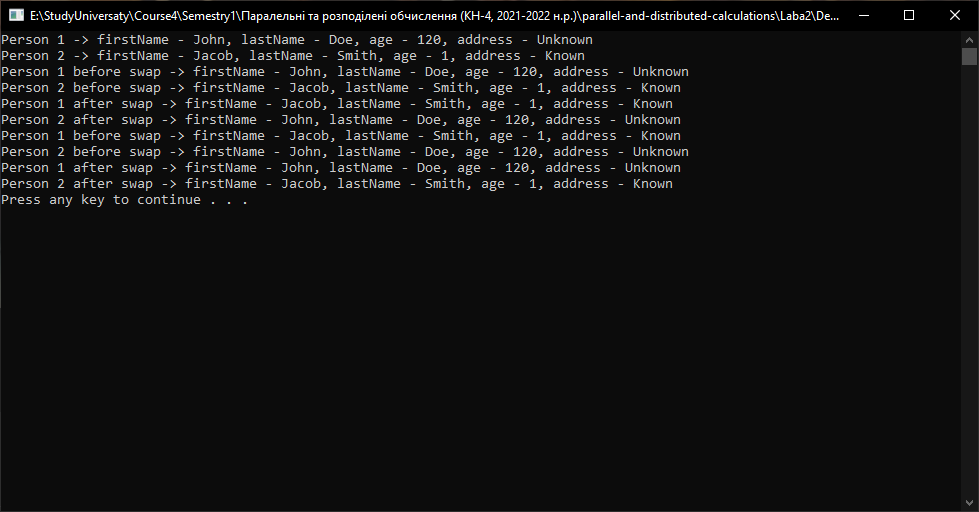
t3.join();

system("pause");

return 0;

}

**Результати виконання роботи**:



3)

**Код:**

#include <iostream>

#include <thread>

#include <mutex>

#include <Windows.h>

#include <list>

#include <algorithm>

using namespace std;

class someData {

public:

string firstName;

string lastName;

string address;

int age;

};

class exchangePerson {

public:

someData data;

mutex myMutex;

static void JohnDoe(exchangePerson& person) {

lock\_guard<mutex> guard(person.myMutex);

person.data.firstName = "John";

person.data.lastName = "Doe";

person.data.address = "Unknown";

person.data.age = 120;

}

static void JacobSmith(exchangePerson& person) {

lock\_guard<mutex> guard(person.myMutex);

person.data.firstName = "Jacob";

person.data.lastName = "Smith";

person.data.address = "Known";

person.data.age = 1;

}

static void Swap(exchangePerson& p1, exchangePerson& p2) {

Show(p1, "Person 1 before swap");

Show(p2, "Person 2 before swap");

if (!AreEqual(p1, p2)){

unique\_lock<mutex> lock1(p1.myMutex, defer\_lock);

unique\_lock<mutex> lock2(p2.myMutex, defer\_lock);

lock(lock1, lock2);

exchangePerson newPerson;

newPerson.data.firstName = p1.data.firstName;

newPerson.data.lastName = p1.data.lastName;

newPerson.data.address = p1.data.address;

newPerson.data.age = p1.data.age;

p1.data.firstName = p2.data.firstName;

p1.data.lastName = p2.data.lastName;

p1.data.address = p2.data.address;

p1.data.age = p2.data.age;

p2.data.firstName = newPerson.data.firstName;

p2.data.lastName = newPerson.data.lastName;

p2.data.address = newPerson.data.address;

p2.data.age = newPerson.data.age;

Show(p1, "Person 1 after swap");

Show(p2, "Person 2 after swap");

}

}

static void Show(exchangePerson& person, string text) {

cout << text << " -> firstName - " << person.data.firstName << ", lastName - " << person.data.lastName << ", age - " << person.data.age << ", address - " << person.data.address << "\n";

}

static bool AreEqual(const exchangePerson& a, const exchangePerson& b){

return addressof(a) == addressof(b);

}

};

void JohnDoe(exchangePerson& person){

person.JohnDoe(person);

person.Show(person, "Person 1");

}

void JacobSmith(exchangePerson& person){

person.JacobSmith(person);

person.Show(person, "Person 2");

}

void Swap(exchangePerson& person1, exchangePerson& person2){

person1.Swap(person1, person2);

person2.Swap(person1, person2);

}

int main() {

exchangePerson person1;

exchangePerson person2;

thread t1(JohnDoe, ref(person1));

t1.join();

thread t2(JacobSmith, ref(person2));

t2.join();

thread t3(Swap, ref(person1), ref(person2));

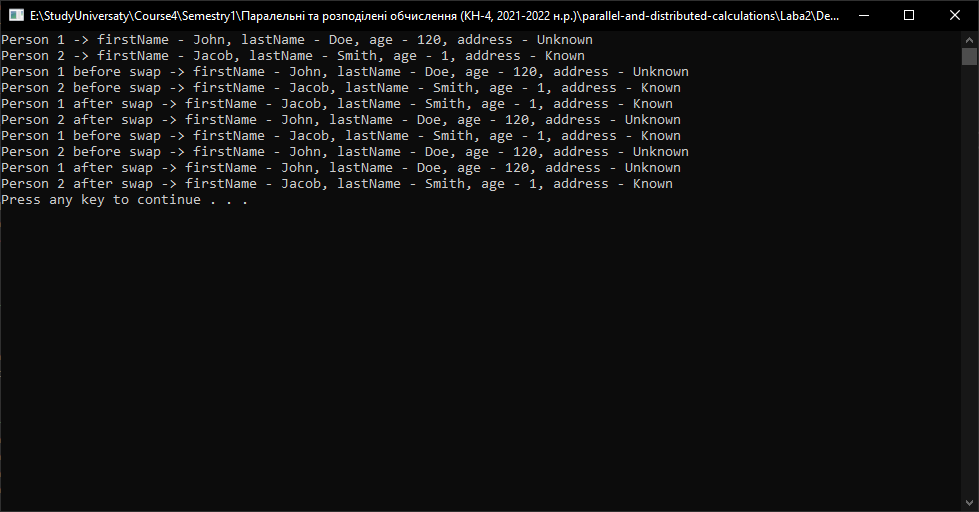
t3.join();

system("pause");

return 0;

}

**Результати виконання роботи:**



Різниця між lock\_guard і unique\_lock в тому, що можна як заблокувати так і розблокувати std :: unique\_lock. Тоді, як std :: lock\_guard буде заблокований тільки один раз при білді і розблоковано при знищенні.

Шаблон lock\_guard використовують, коли об'єкт створений, тоді він намагається отримати м'ютекс. Шаблон unique\_lock підтримує відкладене блокування, тимчасове заблокування, рекурсивне блокування і використання умовних змінних.

**Висновок:** на лабораторній роботі я навчився застосовувати шаблони стандартної бібліотеки на  основі м’ютексів для ізолювання доступу потоків до розділюваних даних. Мої вихідні файли з проектом знаходяться за адресою: <https://github.com/PavloSeniv/parallel-and-distributed-calculations/tree/main/Laba2>