МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ  
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники  
Кафедра программного обеспечения информационных систем и технологий

Отчет по лабораторной работе №13по дисциплине: «Объектно-ориентированное программирование и проектирование»  
  
на тему: «Многопоточность. Синхронизация потоков. Мьютексы»

|  |  |  |
| --- | --- | --- |
| Выполнил: | ст. гр. 10702423 | Якубович И.М. |
| Принял: | доцент | Барков А.В. |

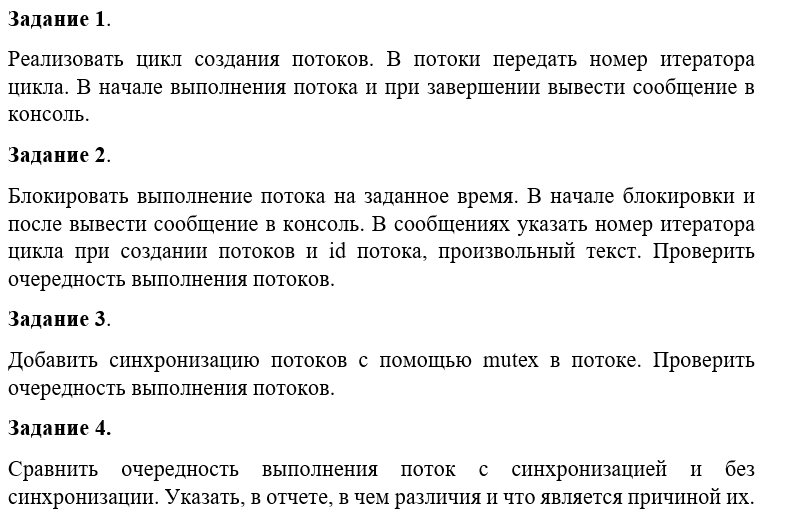
Минск 2025

**Цель работы**

Получить практические навыки синхронизации потоков.

## ВВЕДЕНИЕ

Одной из важных задач при разработке многопоточных программ является организация синхронизации потоков для корректного доступа к разделяемым данным. Без правильной синхронизации возможно возникновение ошибок, таких как гонка данных, нарушение целостности информации или неопределённое поведение программы.



**Код программы**

**Задание 1: main.cpp**

/\*

Задание 1.

Реализовать цикл создания потоков.

В потоки передать номер итератора цикла.

В начале выполнения потока и при завершении вывести сообщение в консоль.

\*/

#include <iostream>

#include <thread>

#include <vector>

void threadTask(int id)

{

*std*::*cout* << "Thread " << id << " started.\n";

*std*::*cout* << "Thread " << id << " finished.\n";

}

int main() {

const int numThreads = 5;

*std*::*vector*<*std*::*thread*> threads;

for (int i = 0; i < numThreads; ++i) {

threads.*emplace\_back*(threadTask, i);

}

for (auto& t : threads) {

t.*join*();

}

return 0;

}

**Задание 2: main.cpp**

/\*

Задание 2.

Блокировать выполнение потока на заданное время.

В начале блокировки и после вывести сообщение в консоль.

В сообщениях указать номер итератора цикла при создании потоков и id потока, произвольный текст.

Проверить очередность выполнения потоков.

\*/

#include <iostream>

#include <thread>

#include <vector>

#include <chrono>

void threadTask(int id) {

*std*::*cout* << "Thread " << id << " started. ID: " << *std*::*this\_thread*::*get\_id*() << "\n";

*std*::*this\_thread*::*sleep\_for*(*std*::*chrono*::*seconds*(2));

*std*::*cout* << "Thread " << id << " resumed. ID: " << *std*::*this\_thread*::*get\_id*() << "\n";

}

int main() {

const int numThreads = 5;

*std*::*vector*<*std*::*thread*> threads;

for (int i = 0; i < numThreads; ++i) {

threads.*emplace\_back*(threadTask, i);

}

for (auto& t : threads) {

t.*join*();

}

return 0;

}

**Задание 3: main.cpp**

/\*

Задание 3.

Добавить синхронизацию потоков с помощью mutex в потоке.

Проверить очередность выполнения потоков.

\*/

#include <iostream>

#include <thread>

#include <vector>

#include <chrono>

#include <mutex>

*std*::*mutex* consoleMutex;

void threadTask(int id) {

*std*::*lock\_guard*<*std*::*mutex*> lock(consoleMutex);

*std*::*cout* << "Thread " << id << " started. ID: " << *std*::*this\_thread*::*get\_id*() << "\n";

*std*::*this\_thread*::*sleep\_for*(*std*::*chrono*::*seconds*(2));

*std*::*cout* << "Thread " << id << " resumed. ID: " << *std*::*this\_thread*::*get\_id*() << "\n";

}

int main() {

const int numThreads = 5;

*std*::*vector*<*std*::*thread*> threads;

for (int i = 0; i < numThreads; ++i) {

threads.*emplace\_back*(threadTask, i);

}

for (auto& t : threads) {

t.*join*();

}

return 0;

}

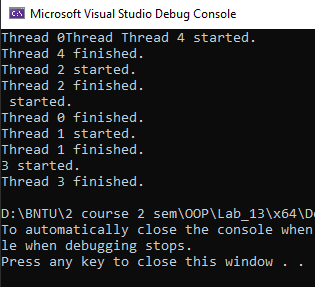
**Задание 4:**

Когда потоки работают без использования мьютекса, они имеют доступ к консоли одновременно. Это приводит к тому, что сообщения от разных потоков могут:

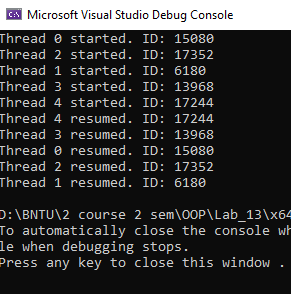
* Перемешиваться,
* Частично накладываться друг на друга,
* Выводиться в произвольном порядке, который зависит от планировщика потоков операционной системы.

Причина:  
Потоки выполняются независимо друг от друга и одновременно обращаются к стандартному выводу (std::cout), который сам по себе не потокобезопасен. Планировщик ОС самостоятельно решает, какой поток когда получает доступ к процессорному времени, поэтому порядок сообщений нельзя заранее предсказать.

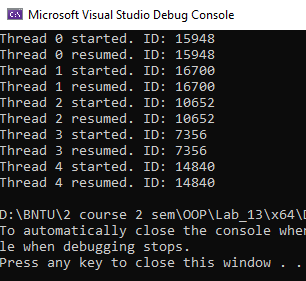
**Тестирование программы:**



*Рисунок 1. Задание 1*



*Рисунок 2. Задание 2*



*Рисунок 3. Задание 3*

**ВЫВОД**

В ходе выполнения лабораторной работы была реализована программа, работающая с применением std::mutex, std::thread. Работа продемонстрировала основные принципы эффективной обработки данных в многопоточной среде:

* Было освоено создание и управление потоками с помощью стандартной библиотеки std::thread.
* Применено деление массива на чанки для равномерного распределения нагрузки между потоками.
* Для защиты общих ресурсов при выводе информации в консоль использован мьютекс std::mutex, что позволило избежать конфликтов при параллельной записи.
* Показано, как можно реализовать параллельное применение простой операции преобразования (умножение на 2) к каждому элементу массива.