# **Лабораторная работа 2: Использование мьютексов в многопоточном программировании**

**Цели работы:**

* Изучить работу с мьютексами в контексте многопоточности.
* Понять важность синхронизации потоков для обеспечения целостности данных.
* Научиться создавать и использовать мьютексы для предотвращения гонок данных.

#include <windows.h>

#include <iostream>

#include <mutex>

std::mutex mtx; // Объявление мьютекса для синхронизации потоков

// Функция, которая будет выполняться в потоках

DWORD WINAPI ThreadFunction(LPVOID lpParam) {

// Захват мьютекса для обеспечения эксклюзивного доступа к ресурсу

mtx.lock();

std::cout << "Поток с ID: " << GetCurrentThreadId() << " выполняется." << std::endl;

// Освобождение мьютекса

mtx.unlock();

return 0;

}

int main() {

setlocale(0, "");

// Установка кодировки консоли на UTF-8 для корректного отображения символов

SetConsoleOutputCP(CP\_UTF8);

HANDLE hThreads[5]; // Массив дескрипторов потоков

// Создание 5 потоков

for (int i = 0; i < 5; i++) {

hThreads[i] = CreateThread(NULL, 0, ThreadFunction, NULL, 0, NULL);

}

// Ожидание завершения всех потоков

WaitForMultipleObjects(5, hThreads, TRUE, INFINITE);

// Закрытие дескрипторов потоков

for (int i = 0; i < 5; i++) {

CloseHandle(hThreads[i]);

}

std::cout << "Все потоки завершены." << std::endl; // Сообщение о завершении всех потоков

return 0;

}

Результат выполнения программы представлен в соответствие с рисунком 1.

# 

Рисунок 1 – Выполнение программы

# **Контрольные вопросы**

1. **Что такое мьютекс и для чего он используется в многопоточном программировании?**

Мьютекс (от англ. mutual exclusion) — это объект синхронизации, который используется для управления доступом к общим ресурсам в рамках многопоточного приложения. Он позволяет только одному потоку в определённый момент времени захватывать доступ к защищённому ресурсу, предотвращая возможность гонок данных.

1. **Как мьютексы помогают избежать гонок данных между потоками?**

Мьютексы позволяют потоку захватывать оборачиваемый ресурс и блокировать доступ к нему для других потоков. Когда один поток захватывает мьютекс, другие потоки, пытающиеся его захватить, будут блокированы до тех пор, пока мьютекс не будет освобождён. Это обеспечивает последовательность выполнения кода, который работает с общими данными, тем самым предотвращая гонки данных.

1. **В чем разница между мьютексами и семафорами?**

Основное различие заключается в том, что мьютекс используется для блокировки доступа к единственному ресурсу, обеспечивая его эксклюзивное использование. Семафор, с другой стороны, управляет доступом к пулу ресурсов; он может разрешать доступ нескольким потокам одновременно (например, если семафор имеет счётчик, позволяющий нескольким потокам). Мьютексы работают по принципу "захват/освобождение", в то время как семафоры могут иметь значение, большее 1.

1. **Как правильно захватывать и освобождать мьютекс в коде?**

Важно всегда захватывать и освобождать мьютексы правильно. Однако лучше использовать RAII для автоматического освобождения мьютекса:

1. **Что произойдет, если поток попытается захватить мьютекс, который уже был захвачен другим потоком?**

Если поток попытается захватить мьютекс, который уже захвачен другим потоком, он будет заблокирован до тех пор, пока мьютекс не будет освобождён. Это может привести к взаимным блокировкам, если два потока захватят мьютексы, ожидая освобождения друг друга.

1. **Какова роль функции std::lock\_guard в управлении мьютексами?**

std::lock\_guard — это класс из стандартной библиотеки C++, который управляет захватом мьютекса. Он автоматически захватывает мьютекс при создании и освобождает его, когда объект lock\_guard выходит из области видимости. Это помогает предотвратить ошибки, связанные с ручным освобождением мьютекса и уменьшает вероятность утечек.

1. **Какие потенциальные проблемы могут возникнуть при неправильном использовании мьютексов (например, взаимные блокировки)?**

При неправильном использовании мьютексов могут возникнуть следующие проблемы:

* Взаимные блокировки (deadlocks): Два или более потоков блокируют друг друга, и каждая ждет освобождения мьютекса, который уже захвачен другим потоком.
* Проблемы с производительностью: Частое блокирование и разблокирование может снизить производительность системы.
* Потеря сигналов: Если один поток захватит мьютекс, а другой ждет его освобождения, может возникнуть ситуация, когда потоки "застрянут".

1. **Как можно проверить, является ли мьютекс доступным для захвата перед его захватом?**

В C++ стандартной библиотеке нет прямого метода для проверки доступности мьютекса, но можно использовать метод try\_lock мьютекса, который пытается захватить мьютекс без блокировки. Если захват успешен, метод возвращает true, в противном случае — false.