Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  
«Национальный исследовательский университет   
Высшая школа экономики»

*Факультет социально-экономических и компьютерных наук*

Мусихин Данил Михайлович

**Работа с многопоточностью**

*Лабораторная работа №5*

студента образовательной программы «Разработка информационных систем для бизнеса» по направлению подготовки *09.03.04 Программная инженерия*

Руководитель, к.т.н., Доцент кафедры ИТБ.

Л.Н. Лядова

Пермь, 2023 год

**Оглавление**

[1 Постановка задачи 4](#_Toc152166223)

[2 Анализ 5](#_Toc152166224)

[3 Код 9](#_Toc152166241)

# Постановка задачи

Цель работы – изучение возможностей создания многопоточных приложений.

[**Загрузка файла с комментариями по выполнению и примером - файл PDF**](https://edu.hse.ru/pluginfile.php/3191824/mod_assign/intro/%D0%97%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%BF%D0%BE%20%D0%A2%D0%B5%D0%BC%D0%B5%207.%20%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C%20%D0%B8%20%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%B0%D0%BC%D0%B8%20%D0%B8%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D0%B0%D0%BC%D0%B8.pdf).

[**Презентация - конспект лекции раздела 1 с теорией по взаимному исключению и тупикам и описанием задач**](https://edu.hse.ru/pluginfile.php/3191824/mod_assign/intro/%D0%9F%D1%80%D0%B5%D0%B7%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F%20-%20%D0%BA%D0%BE%D0%BD%D1%81%D0%BF%D0%B5%D0%BA%D1%82%20%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B8%20%D1%80%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B0%201%20%D1%81%20%D0%BE%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC%20%D0%B7%D0%B0%D0%B4%D0%B0%D1%87.pdf).

**Рекомендации по выполнению**: при выполнении заданий используйте разделы (<https://learn.microsoft.com/ru-ru/dotnet/standard/threading/using-threads-and-threading>, <https://learn.microsoft.com/ru-ru/dotnet/standard/threading/managed-threading-basics>) – подразделы, посвящённые работе с потоками – справочной системы (или файл PDF с полным руководством, доступные по ссылке: <https://learn.microsoft.com/ru-ru/cpp/standard-library/>).

При выполнении работы необходимо рассмотреть возможности:

1)      управления потоками (создания, приостановки, возобновления, завершения);

2)      синхронизации потоков, управления доступом к ресурсам.

**1 – Задача «писатель-читатель» при работе с единичным буфером.**

Поток-писатель записывает в буфер (в переменную в памяти (1–1) или в файл (1–2) – рассмотрите два варианта) единичную запись. Поток читатель считывает и обрабатывает эту запись. Поток-писатель должен первым получить доступ к буферу и записать в него информацию (единичную запись). Поток-писатель после считывания записи потоком-читателем должен перезаписать запись в буфере – обновить данные;  но поток-писатель не должен стирать и перезаписывать запись, не прочитанную потоком-читателем. Поток-читатель не должен «опережать» поток-писатель и считывать пустой буфер, не должен также дважды считывать одну и ту же запись. Данные, которые записываются в буфер и считываются, выводятся на экран в порядке выполнения записи и чтения.

**2 – Задача «писатель-читатель» при работе с циклическим буфером** (буфер – массив фиксированного размера в памяти, в который записываются записи (например данные приборов учёта с временными метками и показателями).

Поток-писатель и поток-читатель записывают и считывают последовательно записи в массиве. При этом поток-писатель не может стереть и перезаписать запись, которая ещё не прочитана потоком-читателем, а поток-читатель не может считывать не заполненным потоком-писателем записи, а также не может дважды прочитать одну и ту же запись. Данные, которые записываются в буфер и считываются, выводятся на экран в порядке выполнения записи и чтения. После выполнения операций изменение буфера отображается на экране.

**3 – Задача «обедающие философы»** («обедающие мудрецы»).

Потоки-«мудрецы» для выполнения своих функций («обед») нуждаются в двух ресурсах («палочках»). Выполнение функции сопровождается выводом информации на экран и случайной задержкой по времени. Общее число ресурсов («палочек») равно числу «мудрецов». Разработайте приложение, реализовав доступ к ресурсам, требующим взаимного исключения («палочкам»), – их монополизацию на время использования, и исключив возможность попадания потоков в тупик.

**4 – Задача «писатели-читатели» при работе нескольких писателей и нескольких читателей с циклическим буфером, реализованным массивом.**

Должны выполняться все требования задачи 1, потоки-писатели не должны перезаписывать информацию, которую записали сами или другие потоки-писатели, пока она не считана потоками-читателями. Каждая запись должна быть прочитана только одним потоком-читателем и не должна быть стёрта, пока она не прочитана.

**5 – Задача «писатели-читатели» при работе нескольких писателей и нескольких читателей с циклическим буфером,** реализованным массивом фиксированного размера, записи из которого выгружаются в файл для долговременного хранения.

Должны выполняться все требования задачи 4, потоки-писатели не должны перезаписывать информацию, которую записали сами или другие потоки-писатели, пока она не считана потоками-читателями. Каждая запись должна быть прочитана только одним потоком-читателем и не должна быть стёрта, пока она не прочитана. Все записанные данные (записи из буфера) должны быть выгружены в файл в том же порядке, в котором они записывались в буфер.

# Анализ

## Задача 1

Для реализации задачи "писатель-читатель" с использованием единичного буфера в С++, вам нужно использовать многопоточное программирование. В данном случае, можно воспользоваться стандартной библиотекой C++11 и выше, которая предоставляет удобные средства для работы с потоками.

Поток-писатель ждет, пока буфер не станет доступным (dataReady = false), записывает данные, устанавливает флаг готовности (dataReady = true) и оповещает поток-читатель. Поток-читатель ждет, пока данные не станут доступными (dataReady = true), считывает данные, сбрасывает флаг и оповещает поток-писатель. Таким образом, обеспечивается правильная синхронизация и порядок выполнения операций.

## Задача 2

Для решения задачи "писатель-читатель" с циклическим буфером вам снова придется использовать мьютексы и условные переменные для обеспечения синхронизации между потоками. В данном случае, также нужно учитывать, что буфер циклический, и писатель и читатель могут "обогнать" друг друга, если не будет корректного контроля индексов.

Каждый поток-писатель ждет, пока есть место для записи, записывает данные, обновляет индексы, и оповещает поток-читатель. Поток-читатель ждет, пока есть данные для чтения, считывает данные, обновляет индексы, и оповещает поток-писатель.

## Задача 3

Проблема "обедающих философов" известна как классическая проблема синхронизации в многозадачных вычислительных системах. Для решения этой проблемы обычно используются мьютексы и условные переменные для обеспечения взаимного исключения и предотвращения возможности взаимной блокировки (deadlock).

В данном коде каждый философ пытается взять две палочки с использованием std::unique\_lock и std::condition\_variable. Если одна из палочек уже занята другим философом, текущий философ освобождает захваченные палочки и пытается взять их снова после короткой паузы. Это позволяет избежать взаимной блокировки. Обратите внимание, что в данной реализации используется рекурсивное обедание: философы продолжают обедать и думать в бесконечном цикле.

## Задача 4

Для решения задачи "писатели-читатели" с несколькими писателями и читателями, используя циклический буфер, нужно применить средства синхронизации для обеспечения корректного доступа к общим ресурсам. В данном случае, вам пригодятся мьютексы и условные переменные.

В этом примере cvReader используется для сигнализации читателям о наличии данных в буфере, а cvWriter используется для сигнализации писателям о том, что есть место для записи. Кроме того, вводится itemCount, чтобы отслеживать количество элементов в буфере.

## Задача 5

Для решения задачи с несколькими писателями и читателями с циклическим буфером, который выгружается в файл, вам потребуется использовать синхронизацию средствами мьютексов и условных переменных, а также механизм для записи в файл в правильном порядке.В примере ниже используется библиотека **<fstream>** для записи данных в файл.

В данном примере каждый поток-писатель записывает свои данные в буфер и в файл, а каждый поток-читатель считывает все записи из буфера. Вы можете настроить количество читателей и писателей в зависимости от вашего конкретного случая.

# Код

## Задача 1

#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

const int bufferSize = 1;

std::string buffer;

std::mutex mtx;

std::condition\_variable cv;

bool dataReady = false;

// Поток-писатель

void writerThread(const std::string& data) {

    std::unique\_lock<std::mutex> lock(mtx);

    // Ждем, пока буфер не освободится

    cv.wait(lock, [] { return !dataReady; });

    // Записываем данные в буфер

    buffer = data;

    std::cout << "Writer: " << data << std::endl;

    // Устанавливаем флаг готовности данных

    dataReady = true;

    // Оповещаем поток-читатель

    cv.notify\_all();

}

// Поток-читатель

void readerThread() {

    std::unique\_lock<std::mutex> lock(mtx);

    // Ждем, пока данные не будут доступны

    cv.wait(lock, [] { return dataReady; });

    // Считываем данные из буфера

    std::cout << "Reader: " << buffer << std::endl;

    // Сбрасываем флаг готовности данных

    dataReady = false;

    // Оповещаем поток-писатель

    cv.notify\_all();

}

int main() {

    // Пример использования

    std::thread writer(writerThread, "Hello, writer!");

    std::thread reader(readerThread);

    writer.join();

    reader.join();

    return 0;

}

## Задача 2

#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

const int bufferSize = 5;  // Размер циклического буфера

int buffer[bufferSize];

int writeIndex = 0;  // Индекс для записи

int readIndex = 0;   // Индекс для чтения

int itemCount = 0;    // Количество элементов в буфере

std::mutex mtx;

std::condition\_variable cv;

// Поток-писатель

void writerThread(int data) {

    std::unique\_lock<std::mutex> lock(mtx);

    // Ждем, пока есть место для записи

    cv.wait(lock, [] { return itemCount < bufferSize; });

    // Записываем данные в буфер

    buffer[writeIndex] = data;

    std::cout << "Writer: " << data << std::endl;

    // Обновляем индекс для записи и количество элементов

    writeIndex = (writeIndex + 1) % bufferSize;

    itemCount++;

    // Оповещаем поток-читатель

    cv.notify\_all();

}

// Поток-читатель

void readerThread() {

    std::unique\_lock<std::mutex> lock(mtx);

    // Ждем, пока есть данные для чтения

    cv.wait(lock, [] { return itemCount > 0; });

    // Считываем данные из буфера

    int data = buffer[readIndex];

    std::cout << "Reader: " << data << std::endl;

    // Обновляем индекс для чтения и количество элементов

    readIndex = (readIndex + 1) % bufferSize;

    itemCount--;

    // Оповещаем поток-писатель

    cv.notify\_all();

}

int main() {

    // Пример использования

    std::thread writer1(writerThread, 1);

    std::thread reader1(readerThread);

    std::thread writer2(writerThread, 2);

    std::thread reader2(readerThread);

    writer1.join();

    reader1.join();

    writer2.join();

    reader2.join();

    return 0;

}

## Задача 3

#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <vector>

const int numPhilosophers = 5;

std::mutex chopstickMutex[numPhilosophers];

std::condition\_variable cv;

void philosopher(int id) {

    int leftChopstick = id;

    int rightChopstick = (id + 1) % numPhilosophers;

    std::unique\_lock<std::mutex> leftLock(chopstickMutex[leftChopstick]);

    std::unique\_lock<std::mutex> rightLock(chopstickMutex[rightChopstick]);

    // Философ обедает (использует палочки)

    std::cout << "Philosopher " << id << " is eating." << std::endl;

    // Имитация времени обеда (случайная задержка)

    std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

    // Философ заканчивает обед и освобождает палочки

    rightLock.unlock();

    leftLock.unlock();

    // Имитация времени между приемами пищи (случайная задержка)

    std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

    // Философ думает

    std::cout << "Philosopher " << id << " is thinking." << std::endl;

    // Имитация времени размышлений (случайная задержка)

    std::this\_thread::sleep\_for(std::chrono::milliseconds(1000));

    // После размышлений философ вновь голоден и повторяет процесс

    philosopher(id);

}

int main() {

    std::vector<std::thread> philosophers;

    // Создание потоков для философов

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

        philosophers.emplace\_back(philosopher, i);

    }

    // Ожидание завершения всех потоков

    for (auto& thread : philosophers) {

        thread.join();

    }

    return 0;

}

## Задача 4

#include <iostream>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <vector>

#include <string>

const int bufferSize = 5;

std::string buffer[bufferSize];

int writeIndex = 0;

int readIndex = 0;

int itemCount = 0;

std::mutex mtx;

std::condition\_variable cvReader, cvWriter;

void writerThread(int id) {

    for (int i = 0; i < 5; ++i) {

        std::unique\_lock<std::mutex> lock(mtx);

        // Ждем, пока есть место для записи

        cvWriter.wait(lock, [] { return itemCount < bufferSize; });

        // Записываем данные в буфер

        buffer[writeIndex] = "Writer " + std::to\_string(id) + ": Data " + std::to\_string(i);

        std::cout << buffer[writeIndex] << std::endl;

        // Обновляем индекс для записи и количество элементов

        writeIndex = (writeIndex + 1) % bufferSize;

        itemCount++;

        // Оповещаем потоки-читатели

        cvReader.notify\_all();

        cvWriter.notify\_all();

    }

}

void readerThread(int id) {

    for (int i = 0; i < 5; ++i) {

        std::unique\_lock<std::mutex> lock(mtx);

        // Ждем, пока есть данные для чтения

        cvReader.wait(lock, [] { return itemCount > 0; });

        // Считываем данные из буфера

        std::cout << "Reader " << id << ": " << buffer[readIndex] << std::endl;

        // Обновляем индекс для чтения и количество элементов

        readIndex = (readIndex + 1) % bufferSize;

        itemCount--;

        // Оповещаем потоки-писатели

        cvWriter.notify\_all();

        cvReader.notify\_all();

    }

}

int main() {

    std::vector<std::thread> writers, readers;

    // Создание потоков для писателей и читателей

    for (int i = 0; i < 3; ++i) {

        writers.emplace\_back(writerThread, i);

        readers.emplace\_back(readerThread, i);

    }

    // Ожидание завершения всех потоков

    for (auto& writer : writers) {

        writer.join();

    }

    for (auto& reader : readers) {

        reader.join();

    }

    return 0;

}

## Задача 5

#include <iostream>

#include <fstream>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <vector>

#include <string>

const int bufferSize = 5;

std::string buffer[bufferSize];

int writeIndex = 0;

int readIndex = 0;

int itemCount = 0;

std::mutex mtx;

std::condition\_variable cvReader, cvWriter;

std::ofstream outputFile("task5.txt");

void writerThread(int id) {

    for (int i = 0; i < 5; ++i) {

        std::unique\_lock<std::mutex> lock(mtx);

        // Ждем, пока есть место для записи

        cvWriter.wait(lock, [] { return itemCount < bufferSize; });

        // Записываем данные в буфер

        buffer[writeIndex] = "Writer " + std::to\_string(id) + ": Data " + std::to\_string(i);

        std::cout << buffer[writeIndex] << std::endl;

        // Записываем данные в файл

        outputFile << buffer[writeIndex] << std::endl;

        // Обновляем индекс для записи и количество элементов

        writeIndex = (writeIndex + 1) % bufferSize;

        itemCount++;

        // Оповещаем потоки-читатели

        cvReader.notify\_all();

        cvWriter.notify\_all();

    }

}

void readerThread() {

    for (int i = 0; i < 15; ++i) {  // При чтении, каждый читатель считывает все записи из буфера

        std::unique\_lock<std::mutex> lock(mtx);

        // Ждем, пока есть данные для чтения

        cvReader.wait(lock, [] { return itemCount > 0; });

        // Считываем данные из буфера

        std::cout << "Reader: " << buffer[readIndex] << std::endl;

        // Обновляем индекс для чтения и количество элементов

        readIndex = (readIndex + 1) % bufferSize;

        itemCount--;

        // Оповещаем потоки-писатели

        cvWriter.notify\_all();

        cvReader.notify\_all();

    }

}

int main() {

    std::vector<std::thread> writers, readers;

    // Создание потоков для писателей и читателей

    for (int i = 0; i < 3; ++i) {

        writers.emplace\_back(writerThread, i);

    }

    readers.emplace\_back(readerThread);

    // Ожидание завершения всех потоков

    for (auto& writer : writers) {

        writer.join();

    }

    for (auto& reader : readers) {

        reader.join();

    }

    // Закрытие файла после завершения работы

    outputFile.close();

    return 0;

}

# Приложение А. Скриншоты работы программ



Рисунок 1 - 1 задача

Изображение выглядит как Шрифт, текст, снимок экрана, типография

Автоматически созданное описание

Рисунок 2 - 2 задача

Изображение выглядит как текст, снимок экрана, шаблон, ткань

Автоматически созданное описание

Рисунок 3 - 3 задача

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Рисунок 4 - 4 задача

Изображение выглядит как текст, снимок экрана, меню, дизайн

Автоматически созданное описание

Рисунок 5 - 5 задача

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 6- Задача 5. Запись в файле