Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

**. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ (ПОТОКОВ): ВЗАИМНОЕ**

**ИСКЛЮЧЕНИЕ И СИНХРОНИЗАЦИЯ**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи и программа для её решения 3](#_Toc182478681)

[1.1 Постановка задачи 3](#_Toc182478682)

[1.2 Демонстрация работы программы 3](#_Toc182478683)

[Заключение 5](#_Toc182478684)

[Приложение А (справочное) Исходный код  
 (к пункту 1.2) 7](#_Toc182478685)

**1 ПОСТАНОВКА ЗАДАЧИ И ПРОГРАММА ДЛЯ ЕЁ РЕШЕНИЯ**

* 1. **Постановка задачи**

В рамках данной задачи необходимо разработать приложение, демонстрирующее параллельную обработку данных в разделе памяти с использованием механизмов синхронизации. Целью является решение проблемы совместного доступа к критическим ресурсам между несколькими потоками, что позволит избежать коллизий и обеспечить согласованность данных.

Приложение должно реализовать модель «писатели-читатели», где несколько потоков могут одновременно читать данные, но только один поток может их изменять. Это позволит эффективно использовать ресурсы системы, обеспечивая при этом безопасность и целостность данных.

В качестве примера обработки могут быть использованы простые операции, такие как генерация случайных чисел, запись их в разделяемую память и последующее чтение этих данных различными потоками. Основное внимание будет уделено технической реализации, а не качествам математической модели.

Приложение должно быть консольным, с четким отображением работы потоков и их взаимодействия. В результате выполнения задачи будет создана система, позволяющая анализировать корректность работы потоков, выявлять возможные коллизии и оценивать эффективность выбранных механизмов синхронизации.

Для достижения этой цели могут быть использованы следующие функции и механизмы: разделяемая память [1], *shared*\_mutex [2], потоки и синхронизация [3].

**1.2 Демонстрация работы программы**

После компиляции приложения, его можно запустить в консольном режиме. При старте приложение создаст несколько потоков: читателей и писателей, которые будут взаимодействовать с разделяемой памятью, выводя информацию о считывании и записи данных (рисунок 1.1).

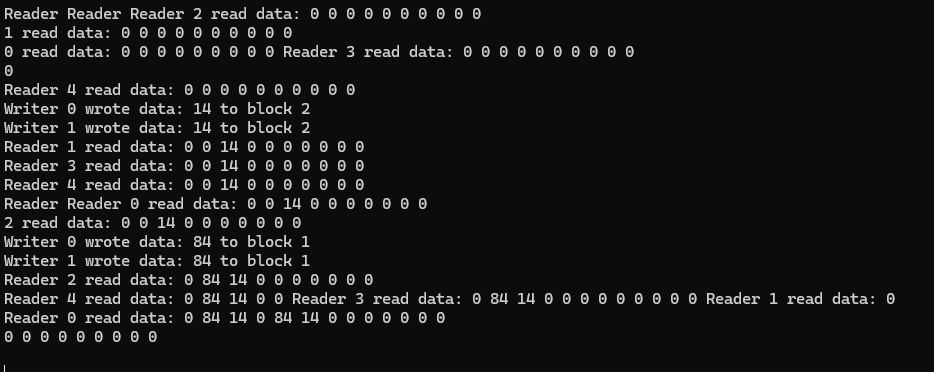


Рисунок 1.1 – Запись и считывание данных

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана программа, демонстрирующая параллельную обработку данных с использованием разделяемой памяти и механизмов синхронизации в модели «писатели-читатели». Основные цели, поставленные в начале работы, были успешно достигнуты.

Лабораторная работа подтвердила, что правильная организация межпроцессного взаимодействия и синхронизации является ключом к успешной реализации многопоточных приложений. Разработанное приложение может служить основой для дальнейших исследований и экспериментов в области параллельной обработки данных.

В будущем можно рассмотреть возможность использования более сложных механизмов синхронизации, таких как условия или барьеры, а также провести эксперименты с различными моделями взаимодействия для углубленного понимания вопросов производительности и масштабируемости многопоточных систем.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] shared\_ptr [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/cpp/memory/shared\_ptr.

[2] shared\_mutex [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/cpp/thread/shared\_mutex.

[3] Concurrency support library [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/cpp/thread.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный код

#include <iostream>

#include <vector>

#include <thread>

#include <mutex>

#include <shared\_mutex>

#include <chrono>

#include <random>

const int NUM\_BLOCKS = 10;

const int NUM\_READERS = 5;

const int NUM\_WRITERS = 2;

struct SharedMemory {

std::vector<int> data;

std::shared\_mutex rw\_mutex;

SharedMemory() : data(NUM\_BLOCKS, 0) {}

};

void reader(SharedMemory& shared\_mem, int id) {

while (true) {

{

std::shared\_lock<std::shared\_mutex> lock(shared\_mem.rw\_mutex);

std::cout << "Reader " << id << " read data: ";

for (const auto& value : shared\_mem.data) {

std::cout << value << " ";

}

std::cout << std::endl;

}

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

}

}

void writer(SharedMemory& shared\_mem, int id) {

std::default\_random\_engine generator;

std::uniform\_int\_distribution<int> distribution(1, 100);

while (true) {

{

std::unique\_lock<std::shared\_mutex> lock(shared\_mem.rw\_mutex);

int block = distribution(generator) % NUM\_BLOCKS;

shared\_mem.data[block] = distribution(generator);

std::cout << "Writer " << id << " wrote data: " << shared\_mem.data[block] << " to block " << block << std::endl;

}

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

}

}

int main() {

SharedMemory shared\_mem;

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

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

for (int i = 0; i < NUM\_READERS; ++i) {

readers.emplace\_back(reader, std::ref(shared\_mem), i);

}

for (int i = 0; i < NUM\_WRITERS; ++i) {

writers.emplace\_back(writer, std::ref(shared\_mem), i);

}

for (auto& t : readers) {

t.join();

}

for (auto& t : writers) {

t.join();

}

return 0;

}