Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

# СИНХРОНИЗАЦИЯ ПОТОКОВ В OPENMP

Отчёт о лабораторной работе № 5 по дисциплине «Параллельное программирование»

Студент гр. 431-3

\_\_\_\_\_\_\_ Д.П. Андреев

«\_\_» \_\_\_\_\_\_\_\_\_ 2024

Проверил

Доцент каф. АСУ, к.т.н

\_\_\_\_\_\_\_ С.М. Алфёров

«\_\_» \_\_\_\_\_\_\_\_\_ 2024

Томск 2024

**1 Цель лабораторной работы**

Цель: освоить методы синхронизации в параллельных программах в задаче Производитель-Потребитель и других задачах, выполняемых на множестве параллельных секций в среде OpenMP.

## 2 Задание

Задание на лабораторную работу: Используя OpenMP написать реализацию защищённого буффера для задачи Производителя-Подребителя. Вариант 9 - Производитель – потребитель. Одиночный буфер. Замок и условная переменная (п.2.3).

### 3 Использованные OpenMP функции

В программе для численного интегрирования были использованы несколько ключевых функций и директив OpenMP, которые обеспечивают параллельное выполнение, управление потоками и сбор результатов вычислений.

1. **omp parallel section**: Директива #pragma omp parallel section указывает компилятору, что секции кода, внутри этой параллельной области, могут быть выполнены параллельно.
2. **omp section**: Директива #pragma omp section Эта директива определяет отдельную секцию в блоке кода. Каждая секция будет выполнена в отдельном потоке, но все секции начинают выполняться одновременно.

Условные переменные и замки использовались из стандартной библиотеки C++23.

**4 Листинг программы**

Main.cpp:

#include <iostream>

#include <thread>

#include <mutex>

#include <atomic>

#include <condition\_variable>

#include <omp.h>

#include <queue>

#include <cstdlib>

#include <chrono>

#include <concepts>

constexpr auto MAX\_ITERATIONS = 10;

inline static auto random\_sleep(int min, int max) -> void {

std::this\_thread::sleep\_for(std::chrono::milliseconds(min + rand() % (max - min + 1)));

}

template<typename T>

concept Numeric = std::integral<T> || std::floating\_point<T>;

struct [[nodiscard]] BufferState {

std::queue<decltype(int())> queue;

std::atomic\_bool buffer\_full = false;

};

std::mutex mtx;

std::condition\_variable cv;

std::atomic\_bool done = false;

namespace mp\_sections {

inline auto producer(Numeric auto max\_iterations, BufferState& state) -> void {

for (auto i = 1; i <= max\_iterations; ++i) {

random\_sleep(1, 100);

std::unique\_lock lock(mtx);

cv.wait(lock, [&state] { return !state.buffer\_full; });

auto value = i; // Выработка значения

state.queue.push(value);

std::cout << "\033[35m-> Producer\033[0m: " << value << " at iteration " << i << "\n";

state.buffer\_full = true;

cv.notify\_one();

}

done = true;

cv.notify\_all();

}

inline auto consumer(Numeric auto max\_iterations, BufferState& state) -> void {

for (auto i = 1; i <= max\_iterations; ++i) {

std::unique\_lock lock(mtx);

cv.wait(lock, [&state] { return state.buffer\_full || done; });

if (!state.queue.empty()) {

auto value = state.queue.front();

state.queue.pop();

std::cout << "\033[36m<- Consumer\033[0m: " << value << " at iteration " << i << "\n";

state.buffer\_full = false;

cv.notify\_one();

}

}

}

} // namespace mp\_sections

// Основная функция

auto main(int argc, char\*\* argv) -> decltype(int()) {

auto iterations = MAX\_ITERATIONS;

if (argc > 1) {

iterations = std::atoi(argv[1]);

}

BufferState buffer\_state{};

// Параллелизация через секции

#pragma omp parallel sections

{

#pragma omp section

{

mp\_sections::producer(iterations, buffer\_state);

}

#pragma omp section

{

mp\_sections::consumer(iterations, buffer\_state);

}

}

return {};

}

### 5 Примеры работы программы

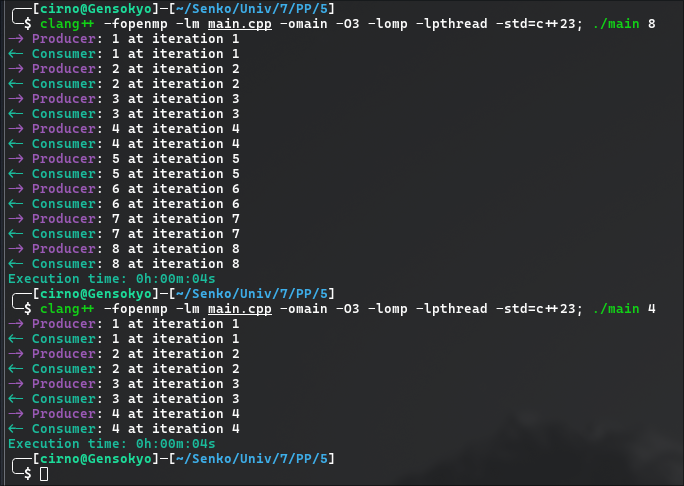
Разберём работу программы. Пример работы изображён на рисунке 5.1.

Рисунок 5.1 — пример работы программы

На рисунке изображён процесс компиляции и работы с защищённым буфером.

### 6 Выводы

Таким образом, я изучил основы потокозащищённости для OpenMP. Я научился применять мьютексы и условные переменные для обеспечения защищённости буфера по заданию.