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 - Читатели-писатели. Приоритет писателей. Два замка и критическая секция (п.3.2).

### 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\_Sync.cpp:

#include <iostream>

#include <fstream>

#include <omp.h>

#include <unistd.h>

#include <thread>

//Вариант № 16: Читатели-писатели. Приоритет писателей. Два замка и критическая секция (п.3.2)

// Как только появился хоть один писатель, никого больше не пускать. Все остальные могут простаивать.

using namespace std;

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

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

//}

int Nrdr = 0; //активные читатели

int info = 0;

ofstream file;

omp\_lock\_t readers; // определяем специальный тип данных под замки

omp\_lock\_t writers;

void Reader(int iter\_num)//Читатель

{

#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

{

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is waiting" << endl;

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is waiting" << endl;

}

usleep(50 + rand() % (1000 - 50 + 1)); // задержка

omp\_set\_lock(&readers); // Блокируем читателей для нормального увеличения счётчика и работы с блоком писателя

Nrdr += 1;

omp\_unset\_lock(&readers); // остальные могут заходить и повышать счётчик

#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

{

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") walked in" << endl;

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") walked in" << endl;

}

usleep(50 + rand() % (1000 - 50 + 1));

#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

{

int my\_info = info;

usleep(50 + rand() % (1000 - 50 + 1));

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is reading data: " << my\_info << "\n";

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is reading data: " << my\_info << endl;

usleep(50 + rand() % (1000 - 50 + 1)); // задержка

}

omp\_set\_lock(&readers); //Также блокируем для уменьшения счётчика

Nrdr -= 1;

omp\_unset\_lock(&readers);

#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

{

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") left." << endl;

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") left." << endl;

}

}

void Writer(int iter\_num)//Писатель

{

// блокируем всех остальных

omp\_set\_lock(&writers);

omp\_set\_lock(&readers);

//#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

// {

// }

#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

{

cout << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") Walked in" << endl;

file << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") Walked in" << endl;

// записываем

int my\_info = info;

usleep(50 + rand() % (1000 - 50 + 1));

file << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is writing data: " << ++my\_info << "\n";

cout << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is writing data: " << my\_info << endl;

info = my\_info;

}

// разблокируем читателей и писателей

omp\_unset\_lock(&readers);

usleep(50 + rand() % (1000 - 50 + 1)); // задержка

omp\_unset\_lock(&writers);

usleep(50 + rand() % (1000 - 50 + 1));

}

int main(int argc, char\* argv[])

{

int n; //Число итераций для читателей и писателей

omp\_init\_lock(&readers); //инициализируем второй замок

omp\_init\_lock(&writers);

file.open("output.txt");

if (argc == 2)

n = atoi(argv[1]);

else {

cout << "Enter num of iters: ";

cin >> n;

}

#pragma omp parallel num\_threads(8)

{

#pragma omp sections nowait

{

#pragma omp section

{

for (int i = 0; i < n; i++)

Writer(i);

}

#pragma omp section

{

for (int i = 0; i < n; i++)

Reader(i);

}

#pragma omp section

{

for (int i = 0; i < n; i++)

Writer(i);

}

#pragma omp section

{

for (int i = 0; i < n; i++)

Reader(i);

}

}

}

file.close();

cout << "End!\n";

}

Main\_NoSync.cpp:

#include <iostream>

#include <fstream>

#include <omp.h>

#include <unistd.h>

#include <thread>

//Вариант № 16: Читатели-писатели. Приоритет писателей. Два замка и критическая секция (п.3.2)

// Как только появился хоть один писатель, никого больше не пускать. Все остальные могут простаивать.

using namespace std;

int Nrdr = 0;

int info = 0;

ofstream file;

//omp\_lock\_t readers; // определяем специальный тип данных под замки

//omp\_lock\_t writers;

void Reader(int iter\_num)

{

//#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

//{

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is waiting" << endl;

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is waiting" << endl;

//}

//Sleep(300); // задержка

//omp\_set\_lock(&readers); // Блокируем читателей для нормального увеличения счётчика и работы с блоком писателя

Nrdr += 1;

//omp\_unset\_lock(&readers); // остальные могут заходить и повышать счётчик

//#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

//{

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") walked in" << endl;

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") walked in" << endl;

//}

usleep(50 + rand() % (1000 - 50 + 1));

//#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

//{

int my\_info = info;

usleep(50 + rand() % (1000 - 50 + 1));

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is reading data: " << my\_info << "\n";

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is reading data: " << my\_info << endl;

//Sleep(300); // задержка

//}

//omp\_set\_lock(&readers); //Также блокируем для уменьшения счётчика

Nrdr -= 1;

//omp\_unset\_lock(&readers);

//#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

//{

cout << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") left." << endl;

file << "Reader " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") left." << endl;

//}

}

void Writer(int iter\_num)

{

// блокируем всех остальных

//omp\_set\_lock(&writers);

//omp\_set\_lock(&readers);

//#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

//{

cout << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") Walked in" << endl;

file << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") Walked in" << endl;

//}

//#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все присутствующие здесь потоки

//{

int my\_info = info;

usleep(50 + rand() % (1000 - 50 + 1));

// записываем

file << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is writing data: " << ++my\_info << "\n";

cout << "Writer " << omp\_get\_thread\_num() << " (iter: " << iter\_num << ") is writing data: " << my\_info << endl;

info = my\_info;

//}

// разблокируем читателей и писателей

//omp\_unset\_lock(&readers);

//Sleep(300);// задержка

//omp\_unset\_lock(&writers);

}

int main(int argc, char\* argv[])

{

int n; //Число итераций для читателей и писателей

//omp\_init\_lock(&readers); //инициализируем второй замок

//omp\_init\_lock(&writers);

file.open("output.txt");

if (argc == 2)

n = atoi(argv[1]);

else {

cout << "Enter num of iters: ";

cin >> n;

}

#pragma omp parallel num\_threads(8)

{

#pragma omp sections nowait

{

#pragma omp section

{

for (int i = 0; i < n; i++)

Writer(i);

}

#pragma omp section

{

for (int i = 0; i < n; i++)

Reader(i);

}

#pragma omp section

{

for (int i = 0; i < n; i++)

Writer(i);

}

#pragma omp section

{

for (int i = 0; i < n; i++)

Reader(i);

}

}

}

file.close();

cout << "End!\n";

}

### 5 Примеры работы программы

На рисунке 5.1 представлен результат выполнения задачи с использованием синхронизации выполнения потоков. Все запуски

осуществлялись на 8 потоках.

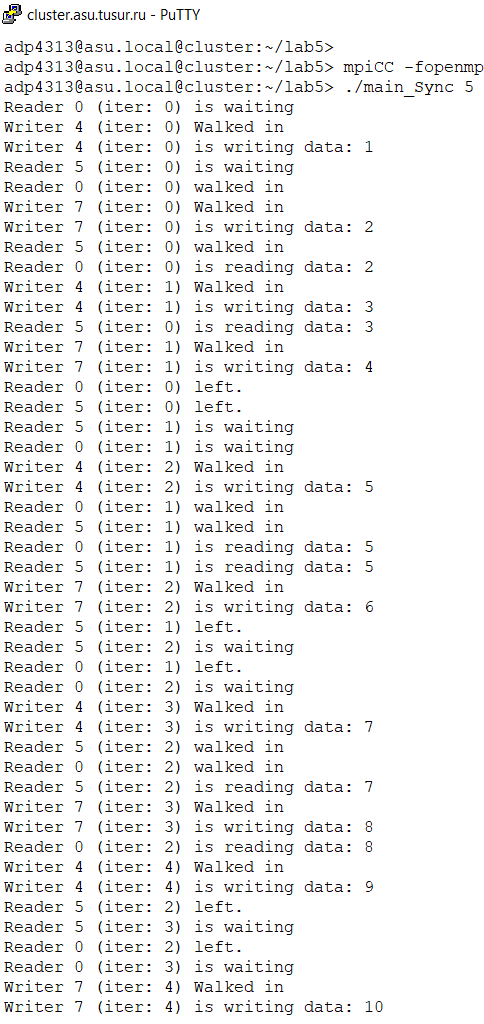


Рисунок 5.1 – Результат работы программы (с синхронизацией)

При запуске режима без синхронизации изменение данных во время работы одного или другого потока с ними – не контролируется. Поэтому возможны ситуации, когда читатель прочитает не обновлённые данные при изменении их писателем. На рисунке 5.2 приведён такой пример, где внизу видна такая ситуация.

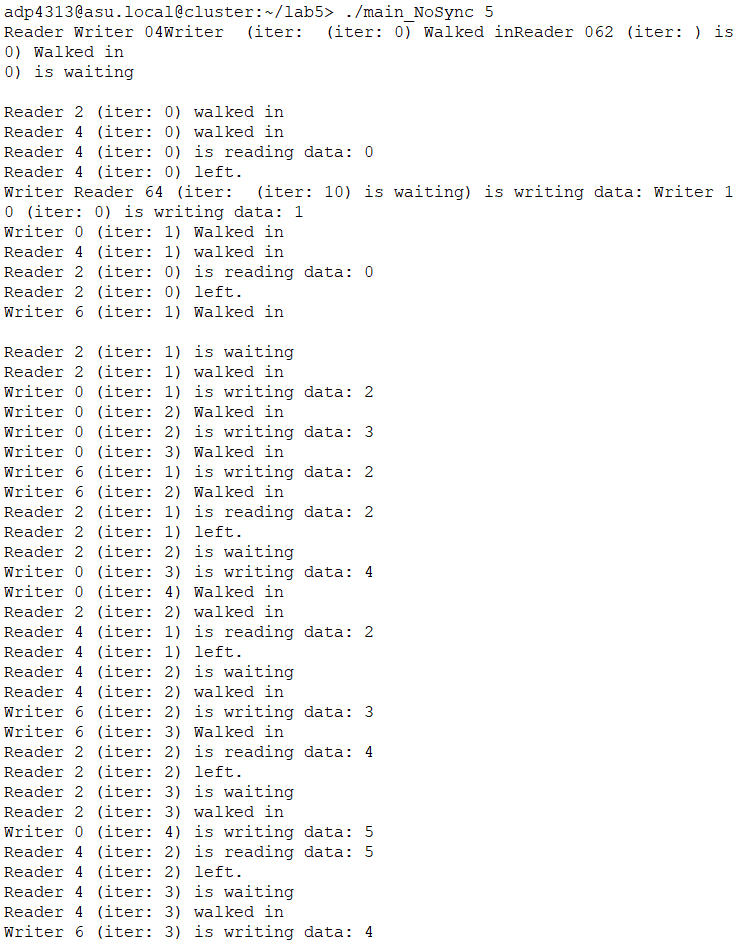


Рисунок 5.2 – Результат работы программы (без синхронизации)

### 6 Выводы

В результате выполнения лабораторной работы были освоены методы синхронизации в параллельных в задаче Производитель-Потребитель и других задачах, выполняемых на множестве параллельных секций в среде OpenMP.