

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 буфера для задачи Производителя-Потребителя. Вариант 9 - Читатели-писатели. Приоритет писателей. Два замка и критическая секция (п.3.2).

3 Используемые 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);
            }

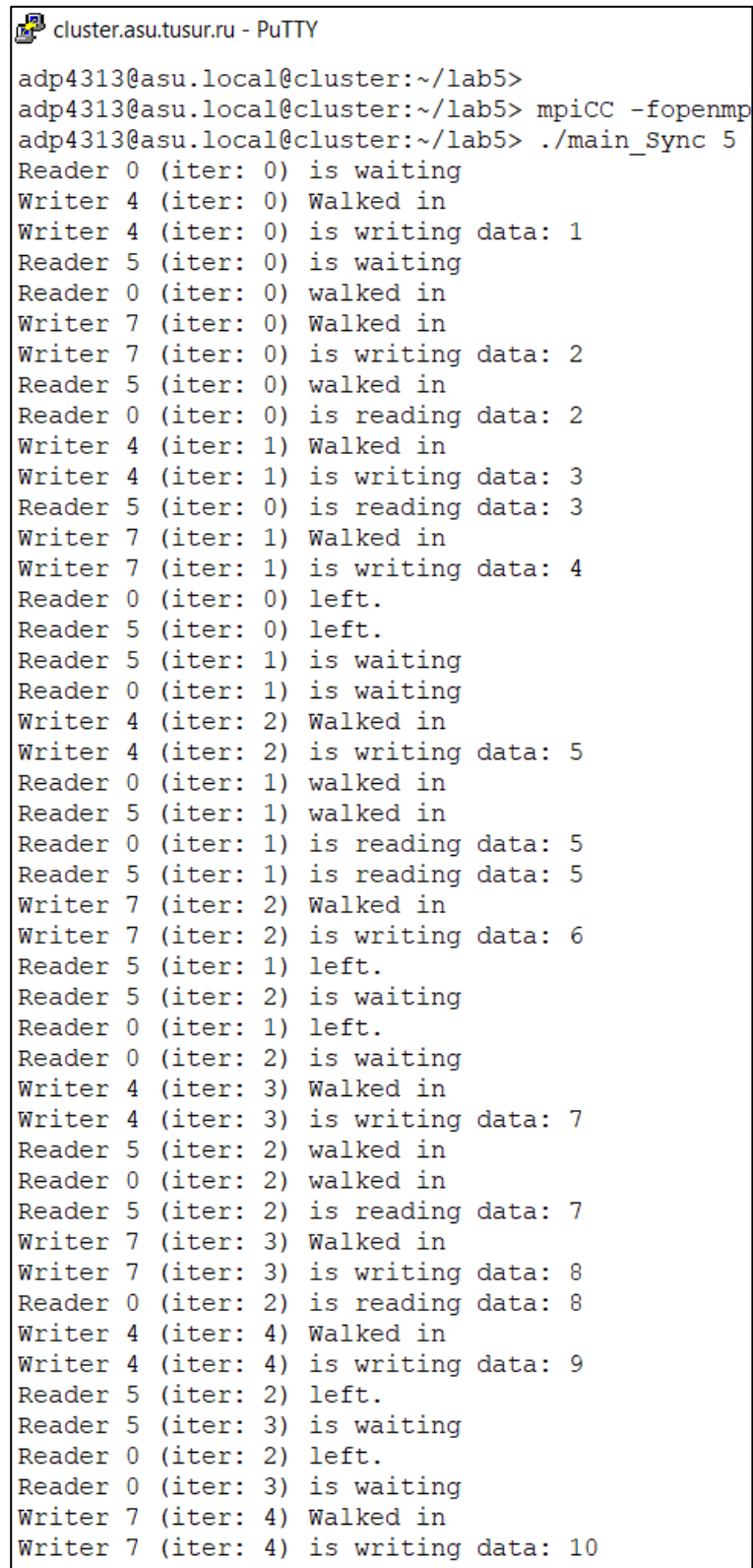
            #pragma omp section
            {
                for (int i = 0; i < n; i++)
                    Writer(i);
            }
        }
    }
}

```

```
        }  
#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.



```
cluster.asu.tusur.ru - PuTTY
adp4313@asu.local@cluster:~/lab5>
adp4313@asu.local@cluster:~/lab5> mpiCC -fopenmp
adp4313@asu.local@cluster:~/lab5> ./main_Sync 5
Reader 0 (iter: 0) is waiting
Writer 4 (iter: 0) Walked in
Writer 4 (iter: 0) is writing data: 1
Reader 5 (iter: 0) is waiting
Reader 0 (iter: 0) walked in
Writer 7 (iter: 0) Walked in
Writer 7 (iter: 0) is writing data: 2
Reader 5 (iter: 0) walked in
Reader 0 (iter: 0) is reading data: 2
Writer 4 (iter: 1) Walked in
Writer 4 (iter: 1) is writing data: 3
Reader 5 (iter: 0) is reading data: 3
Writer 7 (iter: 1) Walked in
Writer 7 (iter: 1) is writing data: 4
Reader 0 (iter: 0) left.
Reader 5 (iter: 0) left.
Reader 5 (iter: 1) is waiting
Reader 0 (iter: 1) is waiting
Writer 4 (iter: 2) Walked in
Writer 4 (iter: 2) is writing data: 5
Reader 0 (iter: 1) walked in
Reader 5 (iter: 1) walked in
Reader 0 (iter: 1) is reading data: 5
Reader 5 (iter: 1) is reading data: 5
Writer 7 (iter: 2) Walked in
Writer 7 (iter: 2) is writing data: 6
Reader 5 (iter: 1) left.
Reader 5 (iter: 2) is waiting
Reader 0 (iter: 1) left.
Reader 0 (iter: 2) is waiting
Writer 4 (iter: 3) Walked in
Writer 4 (iter: 3) is writing data: 7
Reader 5 (iter: 2) walked in
Reader 0 (iter: 2) walked in
Reader 5 (iter: 2) is reading data: 7
Writer 7 (iter: 3) Walked in
Writer 7 (iter: 3) is writing data: 8
Reader 0 (iter: 2) is reading data: 8
Writer 4 (iter: 4) Walked in
Writer 4 (iter: 4) is writing data: 9
Reader 5 (iter: 2) left.
Reader 5 (iter: 3) is waiting
Reader 0 (iter: 2) left.
Reader 0 (iter: 3) is waiting
Writer 7 (iter: 4) Walked in
Writer 7 (iter: 4) is writing data: 10
```

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.

```
adp4313@asu.local@cluster:~/lab5> ./main_NoSync 5
Reader Writer 04Writer (iter: (iter: 0) Walked inReader 062 (iter: ) is
0) Walked in
0) is waiting

Reader 2 (iter: 0) walked in
Reader 4 (iter: 0) walked in
Reader 4 (iter: 0) is reading data: 0
Reader 4 (iter: 0) left.
Writer Reader 64 (iter: (iter: 10) is waiting) is writing data: Writer 1
0 (iter: 0) is writing data: 1
Writer 0 (iter: 1) Walked in
Reader 4 (iter: 1) walked in
Reader 2 (iter: 0) is reading data: 0
Reader 2 (iter: 0) left.
Writer 6 (iter: 1) Walked in

Reader 2 (iter: 1) is waiting
Reader 2 (iter: 1) walked in
Writer 0 (iter: 1) is writing data: 2
Writer 0 (iter: 2) Walked in
Writer 0 (iter: 2) is writing data: 3
Writer 0 (iter: 3) Walked in
Writer 6 (iter: 1) is writing data: 2
Writer 6 (iter: 2) Walked in
Reader 2 (iter: 1) is reading data: 2
Reader 2 (iter: 1) left.
Reader 2 (iter: 2) is waiting
Writer 0 (iter: 3) is writing data: 4
Writer 0 (iter: 4) Walked in
Reader 2 (iter: 2) walked in
Reader 4 (iter: 1) is reading data: 2
Reader 4 (iter: 1) left.
Reader 4 (iter: 2) is waiting
Reader 4 (iter: 2) walked in
Writer 6 (iter: 2) is writing data: 3
Writer 6 (iter: 3) Walked in
Reader 2 (iter: 2) is reading data: 4
Reader 2 (iter: 2) left.
Reader 2 (iter: 3) is waiting
Reader 2 (iter: 3) walked in
Writer 0 (iter: 4) is writing data: 5
Reader 4 (iter: 2) is reading data: 5
Reader 4 (iter: 2) left.
Reader 4 (iter: 3) is waiting
Reader 4 (iter: 3) walked in
Writer 6 (iter: 3) is writing data: 4
```

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.