

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Томский государственный университет систем
управления и радиоэлектроники»

Кафедра автоматизированных систем управления (АСУ)

«СИНХРОНИЗАЦИЯ ПОТОКОВ В OPENMP»

отчет по лабораторной работе №5
по дисциплине «Параллельное программирование»

Обучающийся гр. 431-3
_____ В.Е. Бажинов
«20» декабря 2024 г.

Проверил: доцент каф. АСУ, к.т.н.
_____ С.М. Алфёров
«20» декабря 2024 г.

Томск 2024

Оглавление

1	Цель работы.....	3
2	Ход работы	4
3	Текст программы.....	8
4	Результат работы программы	10
5	Вывод	12

1. Цель работы

Освоить методы синхронизации в параллельных в задаче Производитель-Потребитель и других задачах, выполняемых на множестве параллельных секций в среде OpenMP.

2. Ход работы

1. Проанализировать индивидуальное задание и создать макет программы с необходимым числом секций в параллельной области на основе шаблона индивидуального задания.

Вариант №16 – задача Читателей-Писателей (приоритет писателей). Два замка и критическая секция.

Согласно этой задаче, есть один общий ресурс данных, доступ к которому имеют несколько потоков – записывающие информацию и читающие её. Приоритет писателей заключается в том, что как только появляется хотя бы один писатель, никого больше не пускать. Все остальные могут простаивать.

Используются особые типы данных библиотеки `omp.h`: `omp_lock_t`, которые регулируют порядок выполнения задач несколькими потоками. Использованы два замка – один для писателей, который блокирует доступ писателей к ресурсу, и один для читателей, который блокирует уже читателей, пока, например, писатель не завершит работу. Также использована директива `critical`, которая значит, что в один момент времени только один поток имеет доступ к выполнению содержащихся в директиве команд.

2. Алгоритмы секций оформить функциями. Если несколько секций выполняют одинаковый алгоритм, передать в функцию номер секции. Остальные переменные, используемые в функции, сделать глобальными.

Для читателей и писателей были созданы две отдельные функции – `Readers()` и `Writers()` соответственно.

В качестве глобальных переменных используется общий буфер для хранения данных – в данном случае переменная `info`, хранящая целочисленное значение. При доступе к ней писателей, значение переменной «перезаписывается» (увеличивается на единицу). Также общими переменными являются сами замки, доступ к которым должен быть у каждой функции. Ну и,

наконец, выходной файл, в который записывается информация от читателей и от писателей в хронологическом порядке.

3. Ввести в программу и в секции предложенные в задании средства синхронизации.

В задании предложен метод синхронизации с использованием трёх семафоров, представленный на рисунке 2.1. Данный вариант не совсем подходит. Хотя семафоры и замки схожи по функциональному назначению, количество замков ограничено числом 2.

3.2 Синхронизация с приоритетом писателей

```
VAR Nrdr: integer; W,R,S: Semaphore;

procedure READER;
begin
  P(S);
  P(R);
  Nrdr:=Nrdr+1;
  If Nrdr = 1 then P(W);
  V(S);
  V(R);
  Читать данные;
  P(R);
  Nrdr:=Nrdr-1;
  If Nrdr = 0 then V(W);
  V(R);
end;

Begin
Nrdr:=0; W.C:=1; R.C:=1; S.C:=1;
cobegin
  Repeat READER Until FALSE;
  . . .
  Repeat READER Until FALSE;
  Repeat WRITER Until FALSE;
  . . .
  Repeat WRITER Until FALSE;
coend;
end.
```

Рисунок 2.1 – Вариант решения задачи с использованием 3х семафоров

4. Ограничить выполнение программ секций числом итераций. Число итераций вводить с терминала или передать в главную программу через аргумент командной строки.

Данный пункт был выполнен аналогично такому же пункту в 4-й лабораторной работе – была добавлена обработка аргументов, передаваемых в метод `main()` – если количество аргументов равнялось 2, значит введён параметр количества итераций, и далее он запоминается. Иначе программа запрашивает ввод с терминала.

5. При необходимости включить в алгоритмы задержку на случайный интервал времени.

6. В каждой секции выводить сведения о старте и окончании алгоритма с указанием назначения секции, номера итерации, полученного значения и другую необходимую информацию. Все данные вывести в одну строку для удобства обозрения результатов.

Каждый раз во время запуска функций появляются сообщения о начале работы читателей или писателей с соответствующими функциями.

7. В каждой секции выводить сведения о старте и окончании алгоритма с указанием назначения секции, номера итерации, полученного значения и другую необходимую информацию. Все данные вывести в одну строку для удобства обозрения результатов.

8. Получить два варианта программы: без синхронизации и с синхронизацией.

В варианте без синхронизации все функции и переменные, отвечающие за синхронизацию были исключены путём комментирования.

9. Выполнить обе программы с выводом на экран и файл результатов. Сравнить полученные данные.

3. Текст программы

Листинг кода задачи:

```
#include <iostream>
#include <fstream>
#include <omp.h>
#include <Windows.h>

//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;
}
    Sleep(100);
#pragma omp critical //выполняется только одним потоком за раз, чтобы не писали все
присутствующие здесь потоки
{
    file << "Reader " << omp_get_thread_num() << " (iter: " << iter_num << ") is
reading data: " << info << "\n";
    cout << "Reader " << omp_get_thread_num() << " (iter: " << iter_num << ") is
reading data: " << 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
    {
        // записываем
        info += 1;
        file << "Writer " << omp_get_thread_num() << " (iter: " << iter_num << ") is
writing data: " << info << "\n";
        cout << "Writer " << omp_get_thread_num() << " (iter: " << iter_num << ") is
writing data: " << info << endl;
    }
    // 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);
}
}

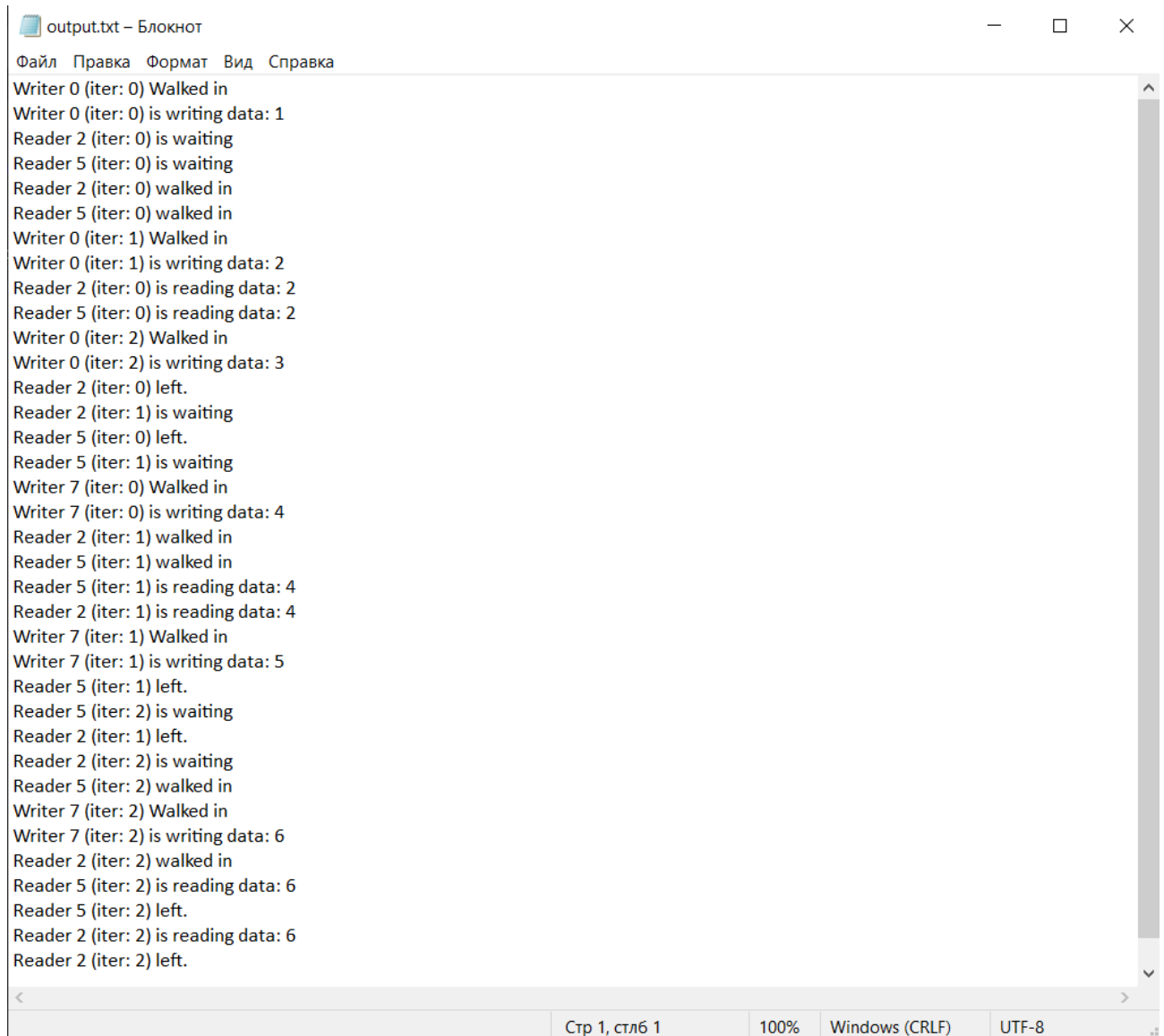
file.close();

cout << "Hello World!\n";
}

```

4. Результат работы программы

На рисунке 4.1 представлен результат выполнения задачи с использованием синхронизации выполнения потоков. Все запуски осуществлялись на 8 потоках.

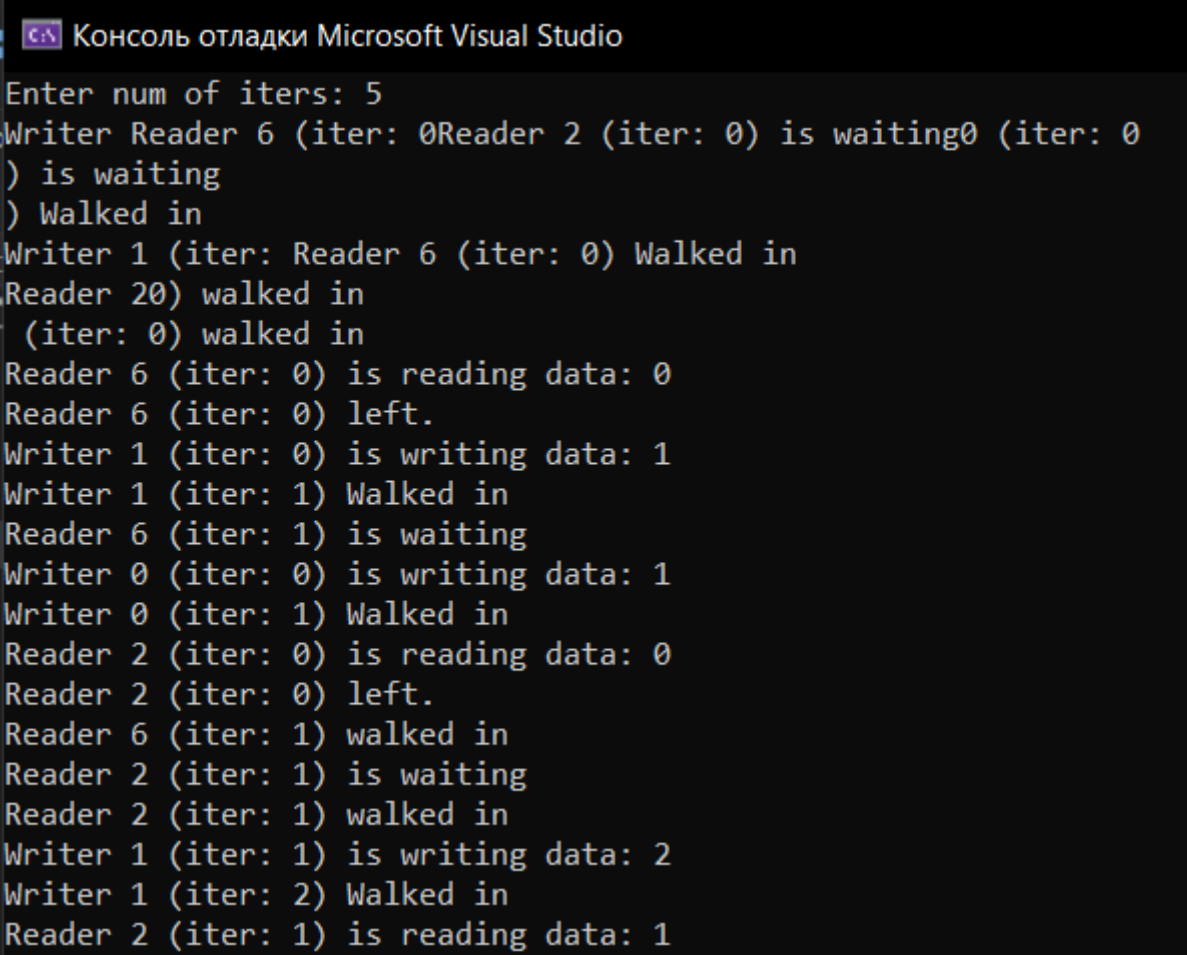


```
output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Writer 0 (iter: 0) Walked in
Writer 0 (iter: 0) is writing data: 1
Reader 2 (iter: 0) is waiting
Reader 5 (iter: 0) is waiting
Reader 2 (iter: 0) walked in
Reader 5 (iter: 0) walked in
Writer 0 (iter: 1) Walked in
Writer 0 (iter: 1) is writing data: 2
Reader 2 (iter: 0) is reading data: 2
Reader 5 (iter: 0) is reading data: 2
Writer 0 (iter: 2) Walked in
Writer 0 (iter: 2) is writing data: 3
Reader 2 (iter: 0) left.
Reader 2 (iter: 1) is waiting
Reader 5 (iter: 0) left.
Reader 5 (iter: 1) is waiting
Writer 7 (iter: 0) Walked in
Writer 7 (iter: 0) is writing data: 4
Reader 2 (iter: 1) walked in
Reader 5 (iter: 1) walked in
Reader 5 (iter: 1) is reading data: 4
Reader 2 (iter: 1) is reading data: 4
Writer 7 (iter: 1) Walked in
Writer 7 (iter: 1) is writing data: 5
Reader 5 (iter: 1) left.
Reader 5 (iter: 2) is waiting
Reader 2 (iter: 1) left.
Reader 2 (iter: 2) is waiting
Reader 5 (iter: 2) walked in
Writer 7 (iter: 2) Walked in
Writer 7 (iter: 2) is writing data: 6
Reader 2 (iter: 2) walked in
Reader 5 (iter: 2) is reading data: 6
Reader 5 (iter: 2) left.
Reader 2 (iter: 2) is reading data: 6
Reader 2 (iter: 2) left.
```

Рисунок 4.1 – 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 необновлённые данные при

изменении их писателем. На рисунке 4.2 приведён такой пример, где внизу видна такая ситуация.



```
Консоль отладки Microsoft Visual Studio
Enter num of iters: 5
Writer Reader 6 (iter: 0) Reader 2 (iter: 0) is waiting 0 (iter: 0
) is waiting
) Walked in
Writer 1 (iter: Reader 6 (iter: 0) Walked in
Reader 2) walked in
(iter: 0) walked in
Reader 6 (iter: 0) is reading data: 0
Reader 6 (iter: 0) left.
Writer 1 (iter: 0) is writing data: 1
Writer 1 (iter: 1) Walked in
Reader 6 (iter: 1) is waiting
Writer 0 (iter: 0) is writing data: 1
Writer 0 (iter: 1) Walked in
Reader 2 (iter: 0) is reading data: 0
Reader 2 (iter: 0) left.
Reader 6 (iter: 1) walked in
Reader 2 (iter: 1) is waiting
Reader 2 (iter: 1) walked in
Writer 1 (iter: 1) is writing data: 2
Writer 1 (iter: 2) Walked in
Reader 2 (iter: 1) is reading data: 1
```

Рисунок 4.2 – Результат работы программы (без синхронизации)

Вывод

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.