инистерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

СИНХРОНИЗАЦИЯ ПОТОКОВ В OpenMP

Лабораторная работа №5 по дисциплине

«Параллельное программирование»

Студент гр. 430-2

_____А.А. Лузинсан

«____»_____2023 г.

Руководитель

Ассистент кафедры АСУ

_____ П.Д. Тихонов

«____»_____2023 г.

Томск 2023

Оглавление

Введение

Цель работы: освоить методы синхронизации в параллельных программах в задаче Читатель-Писатель, выполняемой на множестве параллельных секций в среде OpenMP.

Индивидуальное задание по варианту №6:

Читатели-писатели. Синхронизация с приоритетом писателей.

Шаблон программы:

```
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;

procedure WRITER;
begin
        P(S);
        P(W);
Писать данные;
        V(S);
        V(W);
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.
```

1 Ход работы

1. Проанализировать индивидуальное задание и создать макет программы с необходимым числом секций в параллельной области на основе шаблона индивидуального задания. Алгоритмы секций оформить функциями. Остальные переменные, используемые в функции сделать глобальными.

Для решения индивидуального задания «читатели-писатели» было идентифицировано, что в программе должно быть два класса потоков, которые имеют доступ к некоторому ресурсу. При этом «читатели» - это класс потоков, который может параллельно считывать информацию из общей области памяти, являющейся критическим ресурсом, а «писатели» - это класс потоков, который записывает информацию в эту область памяти, исключая при этом экземпляры собственного класса, а также экземпляры класса «читатели».

В глобальной области при этом объявляются три семафора: W — для управления потоками «писатели», которые получают доступ к разделяемой памяти; R — для взаимоисключения потоков типа «читатели» и S — семафор, обеспечивающий приоритет потокам «писатели». Также, в глобальной области были определены целочисленные переменные: data — представляющая собой критический ресурс и Nrdr — для подсчета активных «читателей».

Далее в программе была определена параллельная область, в которую была вложена директива sections с параметром nowait. Внутри области с sections были определены секции с вызовами функций WRITER() и READER().

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

В функции READER() и WRITER() в соответствии с шаблоном были

добавлены средства синхронизации sem_wait() и sem_post(), которые являются аналогами примитивов P(X) и V(X) семафоров Дейкстры. Данные функции принимают в качестве параметра указатель на объект-семафор, инициализированный вызовом sem_init(). Функция sem_wait(X) проверяет значение заданного семафора X на положительность, уменьшает его на единицу и немедленно возвращает управление процессу. Если значение семафора при вызове функции равно нулю, процесс приостанавливается, до тех пор, пока оно снова не станет больше нуля, после чего значение семафора будет уменьшено на единицу и произойдёт возврат из функции. После завершения работы с семафором поток вызывает функцию sem_post(), которая увеличивает значение семафора на единицу и возобновляет выполнение любых потоков, ожидающих изменения значения семафора. Также в конце программы обязателен вызов функции sem_destroy(), который очищает семафор.

3. Ограничить выполнение программ секций числом итераций . Число итераций вводить с терминала или передать в главную программу через аргумент командной строки. При необходимости включить в алгоритмы задержку на случайный интервал времени: sleep(rand() % t + 1), задержка от 1 до t секунд.

На следующем шаге в каждой секции были оформлены циклы, количество итераций которых задаётся во время запуска главной программы из командной строки.

Помимо этого, возникла необходимость включить в алгоритм задержку в функции READER() и WRITER(). Данная задержка задавалась конструкцией std::this_thread::sleep_for(std::chrono::milliseconds(rand() % x + y)), поэтому для корректной работы программы необходимо подключение библиотек thread и chrono. Промежуток генерации задержки задаётся константно, так как в различных участках было определено различное требуемое поведение.

4. В каждой секции выводить назначение секции, номер итерации, полученного значения и другую необходимую информацию. Все данные вывести в одну строку для удобства обозрения результатов.

В критических секциях была выведена информация о назначении секции, номере потока, а также о количестве активных читателей, номере итерации и считанных данных (в функции READER).

5. Получить два варианта программы: без синхронизации и с синхронизацией. Выполнить обе программы с выводом на экран и файл результатов. Сравнить полученные данные.

Запустив на выполнение программу с синхронизацией и без неё первым делом было замечено, что результаты выполнения без синхронизации не соответствуют ожидаемому поведению работы с критическими ресурсами. Так, доступ к переменной Nrdr получают сразу все читатели, в результате чего число активных пользователей может показываться некорректно. Помимо этого условие с приоритетом читателей не выполняется, так как читатели не ограничены никакими барьерами, в результате чего они смогут считать неактуальные данные и завершить итерацию. При анализе результатов программы с синхронизацией вышеупомянутые проблемы устранены, и писатели актуализируют буфер данных, а читатели, в свободное для писателей время, считывают данные из буфера.

2 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Результат запуска программы с синхронизацией и указанием различного количества итераций, листинг которой представлен в приложении А.1, показан на рисунке 2.1.

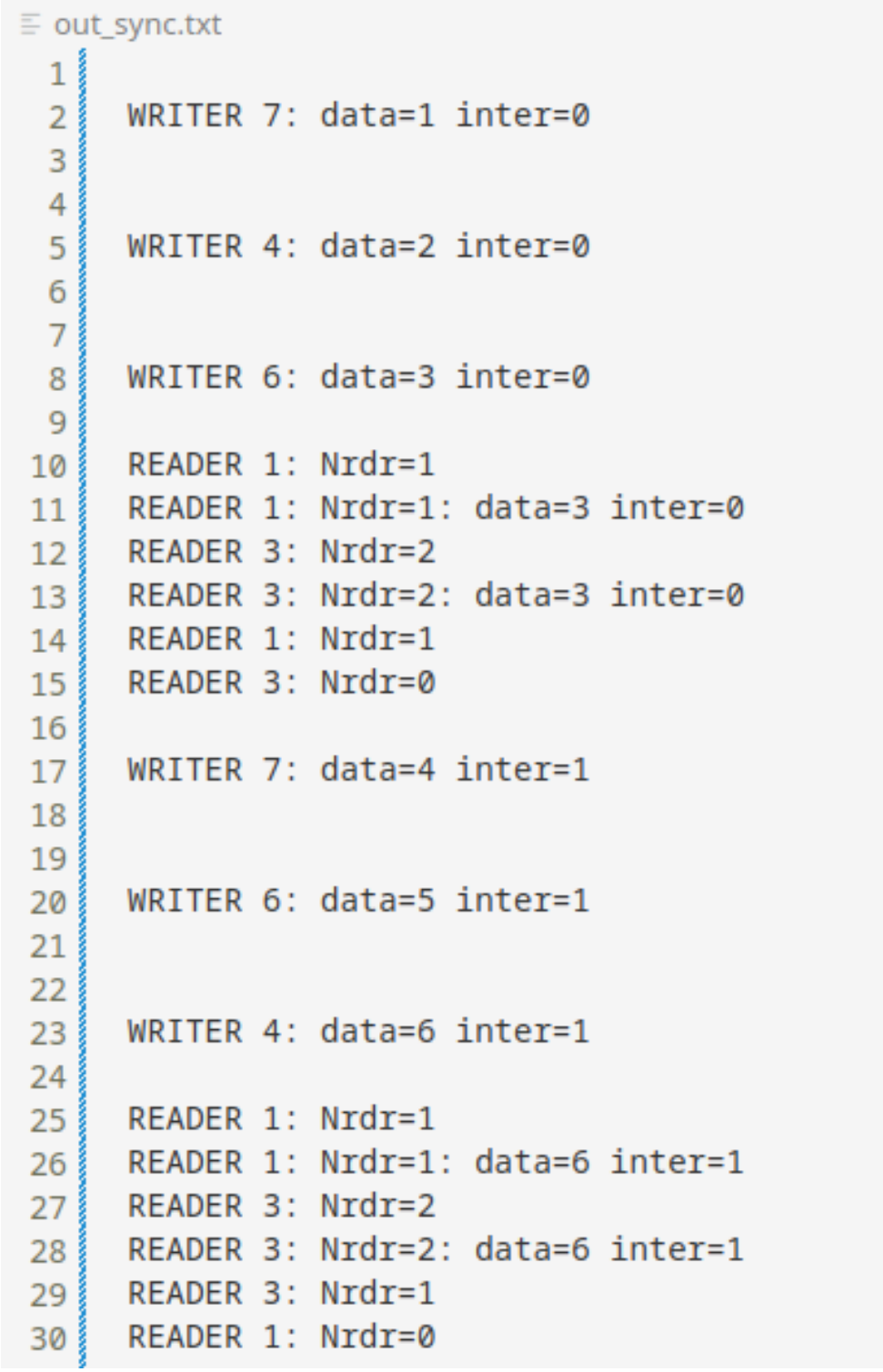

Рисунок 2.1. - Результаты выполнения программы «читатели – писатели» с приоритетом в доступе к критическому ресурсу

Заключение

В результате выполнения лабораторной работы я освоила методы синхронизации в параллельных программах в задаче Читатель-Писатель, выполняемой на множестве параллельных секций в среде OpenMP.

Приложение А
(обязательное)
Листинг программы

Листинг А.1 — Решение задачи «читатели – писатели» с приоритетом в доступе к критическому ресурсу писателей с дополнительным семафором

```
#include <iostream>
#include <omp.h>
#include <semaphore.h>
#include <unistd.h>
#include <chrono>
#include <thread>
#include <fstream>

int Nrdr = 0;
sem_t W, R, S;
int data = 0;
std::ofstream file;

void READER(int n)
{
   std::this_thread::sleep_for(std::chrono::milliseconds(rand() % 1 + 1));
   sem_wait(&S);
   sem_wait(&R);
   Nrdr++;
   #pragma omp critical
   {
        std::cout << "READER " << omp_get_thread_num() << ": Nrdr=" << Nrdr
<< '\n';
      file << "READER " << omp_get_thread_num() << ": Nrdr=" << Nrdr << '\n';
   }
   if (Nrdr == 1) sem_wait(&W);
   sem_post(&S);
   sem_post(&R);
   #pragma omp critical
   {
        std::cout << "READER " << omp_get_thread_num() << ": Nrdr=" << Nrdr
<< ": data=" << data
              << " inter="<<n<<std::endl;
       file << "READER " << omp_get_thread_num() << ": Nrdr=" << Nrdr << ":
data=" << data
              << " inter="<<n<<std::endl;
   }
```

```
    std::this_thread::sleep_for(std::chrono::milliseconds(rand() % 1000 + 300));
    {
        sem_wait(&R);
        Nrdr--;
        #pragma omp critical
        {
            std::cout << "READER "  << omp_get_thread_num() << ": Nrdr=" << Nrdr
<< std::endl;
             file << "READER "  << omp_get_thread_num() << ": Nrdr=" << Nrdr <<
std::endl;
        }
        if (Nrdr == 0) sem_post(&W);
        sem_post(&R);
    }
}

void WRITER(int n)
{
    // std::this_thread::sleep_for(std::chrono::milliseconds(rand() % t + 1));
    sem_wait(&S);
    sem_wait(&W);
    data++;
    #pragma omp critical
    {
         std::cout<< "\nWRITER "  << omp_get_thread_num() << ": data=" << data
<< " inter="<<n<<"\n\n";
        file << "\nWRITER "  << omp_get_thread_num() << ": data=" << data << "
inter="<<n<<"\n\n";
    }
    sem_post(&S);
    sem_post(&W);
    std::this_thread::sleep_for(std::chrono::milliseconds(rand() % 500 + 100));
}

int main(int argc, char *argv[])
{
    sem_init(&W, 0, 1);
    sem_init(&R, 0, 1);
    sem_init(&S, 0, 1);
    file.open("out_sync.txt");

    int n = std::atoi(argv[1]);
```

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

    file.close();
    sem_destroy(&W);
    sem_destroy(&R);

    return 0;
}
```