МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования «Южно-Уральский государственный университет»

(национальный исследовательский университет)

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

ОТЧЕТ

по практической работе № 1

по дисциплине   
«Теория, методы и средства параллельной обработки информации»

Выполнил:

студент группы КЭ–301

/ Д.А. Иванов /

(подпись)

« » 2024 г.

**Задача:**

На языке С/С++ стандартными средствами используемой системы программирования написать параллельное приложение, создающее 3 дополнительных вычислительных потока – 2 «писателя» и 1 «читатель», выполняющих обмен информацией через буфер длины 10. Для простоты под пересылаемой информацией можно понимать целые числа – 1й и 2й потоки их генерируют и записывают в буфер каждый со своей скоростью, 3й поток читает и выводит на экран.

**Листинг программы:**

#include <iostream>

#include <thread>

#include <mutex>

#include <windows.h>

#include <condition\_variable>

const int BUFFER\_SIZE = 10;

int buffer[BUFFER\_SIZE];

int count = 0;

int next\_index = 0;

int next\_read\_index = 0;

std::mutex mtx;

std::condition\_variable cv\_writer, cv\_reader;

void writer\_func() {

    for (int i = 0; i < 10; i++) {

        std::this\_thread::sleep\_for(std::chrono::seconds(1));

        std::unique\_lock<std::mutex> lock(mtx);

        if (count < BUFFER\_SIZE) {

            cv\_writer.wait(lock, [] { return count < BUFFER\_SIZE; });

            buffer[next\_index] = i;

            next\_index = (next\_index + 1) % BUFFER\_SIZE;

            count++;

            //std::cout << "Write 1" << std::endl;

            lock.unlock();

            cv\_reader.notify\_one();

        }

        else {

            continue;

        }

        std::cout << "Write 1: ";

        for (int j = 0; j < count; j++) {

            int index = (next\_read\_index + j) % BUFFER\_SIZE;

            std::cout << buffer[index] << " ";

        }

        std::cout << std::endl;

    }

}

void writer\_func\_2() {

    for (int i = 0; i < 10; i++) {

        std::this\_thread::sleep\_for(std::chrono::seconds(2));

        std::unique\_lock<std::mutex> lock(mtx);

        if (count < BUFFER\_SIZE) {

            cv\_writer.wait(lock, [] { return count < BUFFER\_SIZE; });

            buffer[next\_index] = i;

            next\_index = (next\_index + 1) % BUFFER\_SIZE;

            count++;

            //std::cout << "Write 2" << std::endl;

            lock.unlock();

            cv\_reader.notify\_one();

        }

        else {

            continue;

        }

        // Отражение содержимого буфера на экран

        std::cout << "Write 2: ";

        for (int j = 0; j < count; j++) {

            int index = (next\_read\_index + j) % BUFFER\_SIZE;

            std::cout << buffer[index] << " ";

        }

        std::cout << std::endl;

    }

}

void reader\_func() {

    for (int i = 0; i < 20; i++) {

        std::this\_thread::sleep\_for(std::chrono::seconds(3));

        std::unique\_lock<std::mutex> lock(mtx);

        if (count > 0) {

            cv\_reader.wait(lock, [] { return count > 0; }); // Ждем, пока в буфере появится записанная ячейка

            int data = buffer[next\_read\_index];

            next\_read\_index = (next\_read\_index + 1) % BUFFER\_SIZE;

            count--;

            //std::cout << "Read" << std::endl;

            lock.unlock();

            cv\_writer.notify\_one(); // Сообщаем о появлении свободной ячейки в буфере

        }

        else {

            continue;

        }

        // Отражение содержимого буфера на экран

        std::cout << "Read: ";

        for (int j = 0; j < count; j++) {

            int index = (next\_read\_index + j) % BUFFER\_SIZE;

            std::cout << buffer[index] << " ";

        }

        std::cout << std::endl;

    }

}

int main() {

    std::thread writer\_thread(writer\_func);

    std::thread writer\_thread\_2(writer\_func\_2);

    std::thread reader\_thread(reader\_func);

    writer\_thread.join();

    writer\_thread\_2.join();

    reader\_thread.join();

    return 0;

}

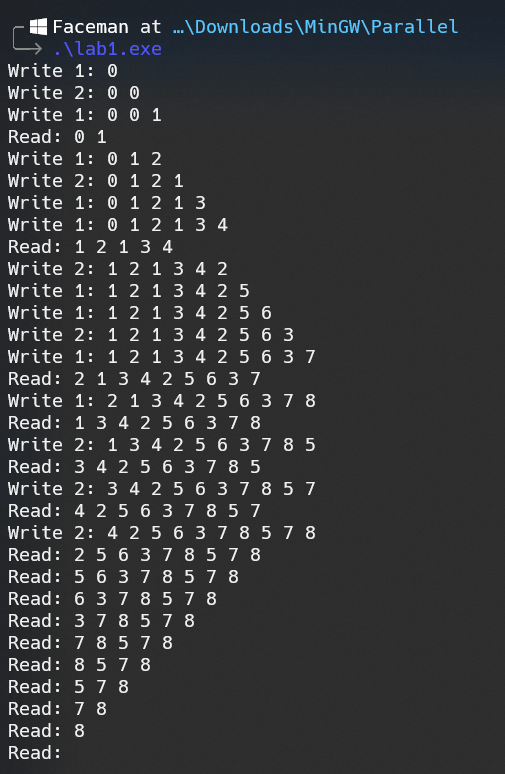


Рисунок 1 – Результат выполнения программы

**Ответы на вопросы:**

1. Привести определения процесса и потока:

Процесс – это выполнение программы в операционной системе. Каждый процесс имеет своё собственное адресное пространство и набор ресурсов. Каждый процесс выполняется независимо от других процессов, что обеспечивает многозадачность в компьютерных системах.

Поток – это наименьшая единица обработки, которая может быть назначена операционной системой для выполнения. Он является отдельной и независимой частью процесса, которая может выполняться параллельно с другими потоками в том же процессе. Потоки позволяют распределить выполнение программы на несколько параллельных потоков, что может улучшить производительность и эффективность выполнения программы.

1. Сколько потоков и в каком порядке создается в ходе работы приложения?

В ходе работы приложения создается 3 потока. Сначала создается поток writer\_thread, который выполняет функцию writer\_func, затем writer\_thread\_2, который выполняет функцию writer\_func\_2 и последним создается reader\_thread для выполнения функции reader\_func. После создания потоков они ожидают завершения работы.

1. Чем определяется порядок выполнения потоков? Какая дисциплина используется?

Порядок выполнения потоков определяется дисциплиной FIFO (First In, First Out). Потоки выполняются в порядке их поступления в очередь. Это означает, что первый поток, который становится готовым к выполнению, будет выполнен первым. Потоки выполняются в том порядке, в котором они были запланированы.

1. Описать все смены состояний потоков в ходе работы приложения.

В начале выполнения программы работает первый писатель, второй писатель и читатель находятся на паузе. Далее первый писатель блокируется, работает второй писатель, читатель на паузе. Позже читатель считывает информацию, пока 2 писателя на паузе. Цикл повторяется.

1. Какова дисциплина обслуживания буфера и почему выбрана именно она?

Дисциплиной обслуживания буфера была выбрана FIFO, так как размер нашего буфера невелик и при такой дисциплине удобно записывать новые данные в конец, а самые старые данные считывать. При переполнении буфера, запись новых данных не будет осуществляться, а при опустошении данных не будет происходить бесконечное считывание.

1. Какие операции доступа к буферу должны синхронизироваться и почему?

К буферу должны синхронизироваться операции записи и операция чтения. Первые чтобы не записывать информацию в одну и ту же ячейку одновременно, вторая чтобы считывать данные, которые пришли раньше всего.

1. Какой механизм выбран для реализации синхронизации и почему? Чем он отличается от классического семафора Дейкстры?

Для реализации синхронизации был выбран мьютекс, с его помощью можно было реализовать блокировку потоков. Различия мьютекса и классического семафора Дейкстры:

- Мьютекс обычно позволяет только одному потоку за раз получить доступ к защищаемому ресурсу, в то время как классический семафор Дейкстры может регулировать доступ нескольких потоков к ресурсу сразу.

- Мьютекс является более простым и обычно более эффективным с точки зрения производительности. Он обеспечивает простой механизм блокировки, который позволяет потоку ждать освобождения ресурса, в то время как семафор Дейкстры требует более сложной логики сигналов и ожиданий.

- Мьютексы обычно реализуются на уровне операционной системы и привязаны к конкретному ресурсу.