МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Параллельные алгоритмы»
ТЕМА: Реализация взаимодействия потоков по шаблону «производитель-потребитель»

Студент гр. 0381 _____ Соколов Д.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2022

Цель работы.

Изучение и практическое применение принципов синхронизации потоков на языке C++, реализация шаблона «производитель-потребитель»

Задание.

На базе лаб. 1 (части 1.2.1 и 1.2.2) реализовать итерационное (потенциально бесконечное) выполнение подготовки, обработки и вывода данных. Обеспечить параллельное выполнение потоков обработки готовой порции данных, подготовки следующей порции данных и вывода предыдущих полученных результатов.

Выполнение работы.

Для представления матриц в программе был использован класс Matrix из предыдущей лабораторной работы. Однако было сделано небольшое добавление в виде поля id и get/set методов для него. Данный функционал используется для корректного вывода в файл и в последствии может быть использован для проверки корректности вычислений (по задумке методы, создающие матрицы заносят в поле id сгенерированных матриц сиды генерации, а при операции суммирования результат суммы наследует id от матриц, учувствовавших в сложении).

Также был написан статический класс MatrixHandler, отвечающий за общий функционал матриц (генерация, сумма двух матриц, вывод матрицы в файл). Все методы и поля класса (переменная класса Generator, также взятая из предыдущей лабораторной работы) статичные для обеспечения общего функционала матриц без необходимости создавать лишнюю сущность лишь с целью выполнения нескольких операций над матрицами. При этом объединение всех методов и необходимых полей в один класс позволяет более удобно хранить и в последствии (возможно) изменять логику операций над матрицами.

Ключевой структурой в данной лабораторной работе является блокирующая очередь (Blocking queue), которая была реализована в виде

шаблонного класса BQueue. В сущности, данная структура является классом-оберткой над стандартной очередью (queue) из библиотеки std. Главной особенностью является тот факт, что в момент совершения операции над очередью доступ к ней блокируется из всех других источников (других потоков). Т.е. доступ остается только у текущего потока, непосредственно осуществляющего операцию.

Очередь BQueue обладает полями:

- max_size – максимальный размер очереди
- size – текущий размер очереди (количество элементов в очереди)
- data – непосредственно очередь (std::queue)
- block – std::mutex, отвечающий за блокировку ресурса
- supply – std::condition_variable, выступающий условием блокировки для операции добавления
- demand – std::condition_variable, выступающий условием блокировки для операции получения объекта из очереди

Очередь поддерживает 2 операции:

- Add(***Template*** item) – добавляет элемент в конец очереди. На время выполнения операции взаимодействие с очередью блокируется при помощи инициализированного внутри функции unique_lock, с передаваемым block. При помощи функции wait() у supply с аргументами unique_lock, и лямбда функцией проверяющей очередь на заполнение. Таким образом обеспечивается ограничение на добавление, в случае если очередь не переполнена. После добавления объекта в очередь data и увеличения размера очереди size, условие demand уведомляется при помощи notify_one() о том, что в очереди появились данные, которые можно из нее получить, после чего происходит разблокировка очереди для добавления.
- Get() – возвращает элемент из начала очереди (т.е. самый старый элемент в очереди). В общем, порядок действий зеркален аналогичному в операции добавления. Доступ также блокируется

при помощи unique_lock, в управление которому передается block. При помощи функции wait() у demand обеспечивается закрытие доступа к очереди на получение. Передаваемая в wait() проверяет очередь на пустоту (отсутствие элементов в очереди). После чего из очереди изымается первый объект и текущий размер очереди size уменьшается. Далее Условие supply уведомляется при помощи notify_one() о том, что в очереди освободилось место, т.к. объект из нее изъяли, после чего происходит разблокировка очереди для получения данных.

Блокировка осуществляется при помощи mutex, unique_lock и condition_variable из библиотеки std. Mutex отвечает за непосредственное управление общим ресурсом. Unique_lock позволяет обеспечить владение ресурсом только для одного потока в определенный момент времени. Condition_variable – условия до (не)выполнения которого необходимо продолжать блокировку.

Таким образом, используя вышеописанные структуры была реализована итеративная, потенциально бесконечная схема «производитель-потребитель» (producer-consumer). Программа разбита на 3 основных потока:

- Генерация матриц - (init)
- Суммирование (разбито на N потоков) – (sum)
- Вывод результата в файл – (out)

Поток init – выступает в роли производителя для sum, sum – выступает в роли производителя для out. Поток sum – является потребителем относительно init и out выступает в роли потребителя относительно sum. Все взаимодействия между данными потоками реализованы при помощи BQueue для матриц и пар матриц.

Демонстрация работы программы:

Программа должна посчитать 5 (iterations) различных матриц размера 4x4, разбивая выполнение каждой суммы на 4 потока. При этом производители-

потребители фиксированы, в соответствии со схемой описанной в предыдущем абзаце.

```
// Matrix dimensions
const int  Config::R = 4;
const int Config::C = 4;

// Defines the maximum number of digits of the matrix values
const int Config::generation_ceiling = 101;

// File output config
const std::string Config::Data_path = "../data/";
const std::string Config::summed = "A+B";

// Multiple-thread sum parameters
const int Config::min_threads = 1;
const int Config::max_threads = 40;
const int Config::summator_threads = 4;

// Number of iterations of performing the operations on matrices
const int Config::iterations = 5;

// Queue size limit
const int Config::max_size = 10;
```

Рисунок 1. Конфигурация программы

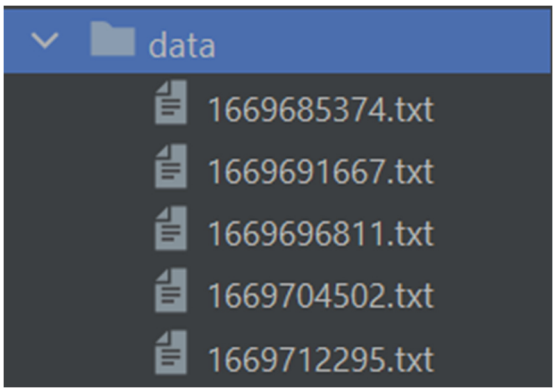

Рисунок 2. Результат работы программы выведен в соответствующую папку

Как было описано ранее, у каждой матрицы есть свой идентификатор (сид ее генерации). Этот идентификатор совпадает у матриц, над которые нужно просуммировать и у результирующей матрицы.

1669685374.txt	A: 4 4 18 47 28 83 93 58 37 71 46 71 23 27 65 81 8 69 B: 4 4 64 62 27 54 53 34 78 100 32 61 32 98 24 8 3 57 A+B: 4 4 82 109 55 137 146 92 115 171 78 132 55 125 89 89 11 126

1669691667.txt	A: 4 4 68 93 88 100 34 94 58 6 42 42 67 13 26 86 83 34 B: 4 4 21 85 84 35 1 6 18 18 7 28 8 87 71 47 6 87 A+B: 4 4 89 178 172 135 35 100 76 24 49 70 75 100 97 133 89 121
1669696811.txt	A: 4 4 74 13 63 41 4 3 34 19 75 98 22 25 41 8 2 69 B: 4 4 74 6 48 84 6 45 21 37 79 59 62 82 64 70 68 70 A+B: 4 4 148 19 111 125 10 48 55 56 154 157 84 107 105 78 70 139

1669704502.txt	A: 4 4 80 55 2 54 61 22 30 51 35 33 91 42 1 69 20 70 B: 4 4 1 56 76 81 93 23 5 3 21 92 74 50 15 24 15 79 A+B: 4 4 81 111 78 135 154 45 35 54 56 125 165 92 16 93 35 149
1669712295.txt	A: 4 4 5 95 57 18 82 39 33 16 65 34 53 78 2 77 95 89 B: 4 4 97 45 55 89 46 29 40 17 71 101 87 59 72 59 5 29 A+B: 4 4 102 140 112 107 128 68 73 33 136 135 140 137 74 136 100 118

Таблица 1. Содержание файлов

Выводы.

В ходе выполнения лабораторной работы была изучена реализация взаимодействия потоков по шаблону «производитель-потребитель», принципы синхронизации потоков на языке C++ с помощью mutex, unique_lock, condition_variable.