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 потокобезопасных структур данных с блокировками

Студент гр. 0303 _____ Бодунов П.А.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Исследовать разницу между "грубой" и "тонкой" блокировками, используя очередь на основе данных по шаблону "производитель-потребитель".

Задание.

Реализовать итерационное (потенциально бесконечное) выполнение подготовки, обработки и вывода данных по шаблону “производитель-потребитель” (на основе лаб. 1 (части 1.2.1 и 1.2.2)).

Обеспечить параллельное выполнение потоков обработки готовой порции данных, подготовки следующей порции данных и вывода предыдущих полученных результатов.

Использовать механизм “условных переменных”.

2.1 Использовать очередь с “грубой” блокировкой.

2.2 Использовать очередь с “тонкой” блокировкой

Сравнить производительность 2.1. и 2.2 в зависимости от количества производителей и потребителей.

Выполнение работы.

Создание матрицы происходит с помощью функции:

void generate_matrix(int rows, int columns) – на вход подается количество строк и столбцов. Элементы матрицы заполняются случайным целочисленными значениями в диапазоне [0, 99].

Создание матриц и добавление массива из созданных матриц в очередь с блокировкой:

void generate_matrices((QueueRude/QueueThin)<vector<vector<int>>*>& qgenerate, int calc_num, int prod_num) – на вход подается ссылка на очередь с "тонкой" или "грубой" блокировкой, количество вычислений и количество производителей. Генерируются 2 матрицы, они добавляются в очередь в виде массива matrices.

Умножение матриц для происходит при помощи функции:

void multiply_matrices((QueueRude/QueueThin)<vector<vector<int>>*>& qgenerate, (QueueRude/QueueThin)<vector<vector<int>>>& qmult, int calc_num, int cons_num) – на вход подаются ссылки на очереди с "тонкой" или "грубой" блокировкой: в qgenerate лежат элементы в виде массивов по 2 элемента, qmult состоит из элементов, хранящих результат перемножения матриц, количество вычислений и количество потребителей.

Запись результирующей матрицы осуществляется при помощи функции:

void write_result((QueueRude/QueueThin)<vector<vector<int>>>& qmult, int calc_num, string filename) – передается ссылка на очередь с "тонкой" или "грубой" блокировкой, количество вычислений и имя файла, в который запишутся все результирующие матрицы.

Были созданы классы очередь с "тонкой" QueueThin и "грубой" QueueRude блокировкой.

Класс QueueRude – описывает очередь с "грубой" блокировкой.

В классе есть поля:

- mutable mutex mut – мьютекс, блокирующий очередь
- queue<T> my_queue – очередь
- condition_variable cv – условная переменная

В классе есть методы:

- void push(T new_value) – добавление элемента new_value в очередь
- void wait_and_pop(T& value) – удаление элемента из очереди и запись значения в value
- bool empty() – проверка очереди на пустоту
- int size() – размер очереди

Класс QueueThin – описывает очередь с "тонкой" блокировкой. В очереди всегда присутствует фиктивный элемент.

В классе была описана структура узла Node:

1) shared_ptr<T> data – указатель на данные для узла

2) unique_ptr<Node> next – указатель на следующий узел

В классе есть поля:

- mutex head_mutex – мьютекс блокирующий голову
- unique_ptr<Node> head – указатель на голову очереди
- mutex tail_mutex – мьютекс блокирующий голову
- Node* tail – указатель на хвост
- condition_variable cv – условная переменная

В классе есть методы:

- Node* get_tail() – возвращает указатель на хвост очереди
- unique_ptr<Node> wait_pop_head(T& value) – удаление элемента из очереди, результат записывается в value
- void wait_and_pop(T& value) – функция вызывающая удаление элемента из очереди, результат записывается в value
- void push(T new_value) – добавление элемента new_value в очередь
- bool empty() – проверка очереди на пустоту

Исследование скорости работы очереди с "грубой" и "тонкой" блокировки

Для этого возьмём постоянный размер матрицы равный 30x30.

Результаты зависимости времени работы программы от количества производителей и потребителей для очереди с "грубой" и "тонкой" блокировками представлены в табл. 1. и 2. соответственно.

Таблица 1 — Зависимость времени работы программы очереди с "грубой" блокировкой

Количество производителей	Количество потребителей	Время работы программы, мкс
1	1	421708

1	20	161188
20	1	452776
7	7	163327
100	100	191883

Таблица 2 — Зависимость времени работы программы очереди с "тонкой" блокировкой

Количество производителей	Количество потребителей	Время работы программы, мкс
1	1	420153
1	20	160926
20	1	442125
7	7	162927
100	100	190010

Исходя из результатов таблиц 1 и 2, время работы "тонкой" блокировки меньше, чем время работы "грубой". Это связано с тем, что очередь с "тонкой" блокировкой блокирует потоки, выполняющие действия с одними и теми же структурами данных, а очередь с "грубой" блокировкой блокирует всю структуру данных.

Выводы.

В процессе выполнения лабораторной работы были изучены и практически реализованы очереди с "тонкой" и "грубой" блокировками на языке C++ для решения задачи производитель-потребитель.