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

ОТЧЕТ

по лабораторной работе №3
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 структуры данных без блокировок, используя очередь на основе данных по шаблону "производитель-потребитель".

Задание.

Выполняется на основе работы 2.

Реализовать очередь, удовлетворяющую lock-free гарантии прогресса.

Протестировать доступ к реализованной структуре данных в случае нескольких потоков производителей и потребителей.

Сравнить производительность с реализациями структур данных из работы 2

Сформулировать инвариант структуры данных.

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.

Класс QueueLockFree – описывает очередь без блокировки. В очереди всегда присутствует фиктивный элемент.

В классе была описана структура узла Node:

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

2) atomic<Node*> next – атомик указателя на следующий узел

В классе есть поля:

- atomic<Node*> head – атомик указателя головы очереди
- atomic<Node*> tail – атомик указателя на хвост
- atomic<Node*> to_be_deleted – атомик указателя на узлы, которые необходимо удалить при возможности
- atomic<unsigned> threads_in_pop – атомик количества потоков, выполняющих удаление элемента из очереди

В классе есть методы:

- shared_ptr<T> pop() – возвращает результат данных головы, удаление головы из очереди
- void push(T val) – добавление элемента со значением val в очередь
- static void delete_nodes(Node* nodes) – удаление списка узлов начиная с nodes
- void try_reclaim(Node* old_head) – попытка удалить old_head, если поток 1, то можно удалить узел на который указывает

old_head, иначе old_head добавляется в список узлов to_be_deleted

- void chain_pending_nodes(Node* first, Node* last) – функция добавляющая очередь с началом first и концом last в to_be_deleted
- void chain_pending_node(Node* n) – добавляет n в to_be_deleted
- void chain_pending_nodes(Node* nodes) – добавляет цепочку с началом в nodes в to_be_deleted

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, а так же очереди без блокировки представлены в табл. 1.

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

Количество производителей	Количество потребителей	Время работы без блокировки, мкс	Время работы "грубой" блокировки, мкс	Время работы "тонкой" блокировки, мкс
1	1	172065	421708	420153
1	20	96439	161188	160926
20	1	244926	452776	442125
7	7	94443	163327	162927
100	100	156143	191883	190010

Исходя из результатов таблицы 1, время работы без блокировки меньше, чем время работы "грубой" и "тонкой". Это связано с тем, что очередь без блокировки не блокирует структуру данных.

Выводы.

В процессе выполнения лабораторной работы были изучены и практически реализована очередь без блокировки на языке C++ для решения задачи производитель-потребитель.