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

Студентка гр. 0303 _____ Костебелова Е. К.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучение способов реализации потокобезопасных структур данных с блокировками.

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.

В ходе выполнения лабораторной работы были реализованы шаблонные классы *threadsafe_queue* и *finegrained_queue* представляющие собой очередь с «грубой» блокировкой и очередь с «тонкой» блокировкой соответственно.

Очередь с «грубой» блокировкой имеет методы добавления элемента *add(T element)* и извлечения элемента *T get()*. Синхронизация обеспечивается с помощью мьютекса и условной переменной. Для извлечения и добавления элемента необходимо заблокировать мьютекс. В случае добавления элемента посылается сигнал условной переменной о том, что очередь не пуста. В случае извлечения элемента необходимо проверить очередь на наличие в ней элементов. Если элементов нет, то нужно дождаться сигнала от условной переменной.

Очередь с «тонкой» блокировкой представляет собой односвязный список, который поддерживает операции добавления элемента *add(T element)* и извлечения элемента *T get()*. Синхронизация обеспечивается использованием двух мьютексов (для головы и для хвоста) и условной переменной. Для добавления элемента в очередь требуется заблокировать мьютекс хвоста, изменить значение текущего хвостового узла и привязать новый хвостовой узел, после чего следует послать сигнал условной переменной. В случае извлечения элемента необходимо заблокировать мьютекс головы очереди, после чего нужно дождаться сигнала от условной переменной, в предикате которой проверяется наличие элементов в очереди. Для проверки нужно заблокировать ещё и мьютекс хвоста. Когда в очереди есть элементы, можно извлекать данные. При этих операциях мьютекс головы списка остаётся заблокированным до конца выполнения извлечения.

Исследование.

Исследование производительности в зависимости от количества производителей и потребителей производилось при помощи запуска программ в течение 100 милисекунд на матрицах 100x100 при разном количестве потоков-производителей и потоков-потребителей.

Результаты для очереди с «грубой» блокировкой представлены в таблице (см. таблица 1).

Таблица 1 – Результаты для очереди с «грубой» блокировкой.

Производители	Потребители	Выполнено умножений
1	1	4
4	4	22
10	10	35
3	6	40
6	3	21

Результаты для очереди с «тонкой» блокировкой представлены в таблице (см. таблица 2).

Таблица 2 – Результаты для очереди с «тонкой» блокировкой.

Производители	Потребители	Выполнено умножений
1	1	9
4	4	34
10	10	200
3	6	80
6	3	29

Из проведённого исследования можно сделать вывод, что программа, которая использует очередь с «тонкой» блокировкой работает быстрее. Это связано с тем, что очередь с «тонкой» блокировкой позволяет уменьшить количество блокировок. Особенно эта разница заметна при большом количестве потоков.

Выводы.

В ходе лабораторной работы были изучены способы реализации потокобезопасных структур данных с блокировками. Были написаны две программы, выполняющие одну и ту же задачу, но использующие разные очереди (с «грубой» и «тонкой» блокировкой). Также было проведено исследование производительности задач в зависимости от количества потоков-производителей и потоков-потребителей. Был сделан вывод о том, что задача, использующая очередь с «тонкой» блокировкой работает быстрее.