**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 Курочкина Е. А.

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 “производительпотребитель” (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.**

Был создан класс Matrix, который принимает на вход размерность матрицы и генерирует случайным образом матрицу данного размера. Для него были перегружены операторы присваивания (с копированием и перемещением), перенаправления в поток и оператор умножения, которая умножает матрицы в 4 потока.

2.1 Для решения данного пункта задачи, был реализован класс Queue. Этот класс – очередь с грубой блокировкой. Данный класс основан на использовании std::queue. Синхронизация потоков реализована путем использования мьютекса и условной переменной.

2.2 Для решения данного пункта задания, был реализован класс FGQueue (Fine grained queue) – очередь с мелкогранулярной блокировкой. Данный класс, отличается от предыдущего тем, что для блокировки поток используются 2 мьютекса (для «головы» и «хвоста» очереди). При вставке нового элемента блокируется только «хвост», а при удалении элемента из очереди – «голова» и «хвост» (для проверки условия в условной переменной).

Для решения каждой подзадачи, используются 2 очереди:

- в первую очередь кладем пары матриц, которые нужно умножить

- во второю очередь – результат умножения матриц для последующей

записи в файл;

В нашей программе «producer» (производитель) генерирует матрицы и кладет их в очередь, а «consumer» (потребитель) - достает эту пару матриц, умножает и кладет в другую очередь.

**Исследование**

Исследуем скорость работы подзадач в зависимости от количества потребителей и количества производителей. Так как, программа потенциально может работать бесконечно, посчитаем количество произведенных и умноженных матриц за 3 секунды.

Таблица 1. Результат работы программы при использовании очереди с

грубой блокировкой.

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

Таблица 2. Результат работы программы при использовании 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 потоковпроизводителей и потоков-потребителей. Был сделан вывод о том, что задача, использующая очередь с «тонкой» блокировкой работает быстрее.