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. 0304 _____ Люлин Д.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучить принципы построения потокобезопасных структур данных с блокировками.

Здание.

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

В отчёте: сравнить производительность 2.1. и 2.2 в зависимости от количества производителей и потребителей.

Выполнение работы.

1. Очередь с «грубой» блокировкой.

Для очереди с блокировками был создан шаблонный класс *ThreadSafeQueue*:

```cpp
/// @brief Потокобезопасная очередь.
/// @details Поддерживает множество производителей и множество потребителей.
/// @tparam T тип элемента очереди.
template < typename T >
class ThreadSafeQueue
{
public:
    /// @brief Завершить работу и все ожидания.
    void finish();

    /// @brief Добавление элемента в очередь.
    /// @param[in] elem добавляемый элемент очереди.
    void push( T&& elem );

    /// @brief Обработка одного элемента очереди, если есть.
    /// @details Когда элементов нет, ждёт, пока появится элементов,
    /// если есть ещё производители.
    /// @param[in] func функция, которая исполнится, если будет получен элемент очереди.
    /// @return true, если элемент был обработан, иначе false.
    bool process( const std::function< void( const T& ) >& func );

private:
    std::queue< T > queue_;                 ///< внутреннее представление очереди.
    std::condition_variable cond_;          ///< условие для ожидания.
    std::mutex mutex_;                      ///< мьютекс для блокировок.
    bool finish_{ false };                  ///< условие окончания работы.
};
```

Сущности, использующие объекты этого класса, разделены на производителей и потребителей. Производители добавляют в очередь элементы, а потребители забирают их из очереди и производят операцию (передаваемую с помощью *std::function*). Все блокировки происходят с помощью одного мьютекса и условной переменной. Мьютекс блокируется в начале каждого метода, а условная переменная – в методе *process()*, для ожидания элементов для обработки.

2. Очередь с «тонкой» блокировкой.

Для очереди с «тонкой» блокировкой был написан шаблонный класс *FineGrainedQueue*:

```cpp
/// @brief Потокобезопасная очередь с тонкими блокировками на односвязном списке.
/// @details Поддерживает множество производителей и множество потребителей.
/// @tparam T тип элемента очереди.
template < typename T >
class FineGrainedQueue
{
public:
    /// @brief Конструктор.
    FineGrainedQueue();

    /// @brief Завершить работу и все ожидания.
    void finish();

    /// @brief Добавление элемента в очередь.
    /// @param[in] elem добавляемый элемент очереди.
    void push( T&& elem );

    /// @brief Обработка одного элемента очереди, если есть.
    /// @details Когда элементов нет, ждёт, пока появится элементов,
    /// если есть ещё производители.
    /// @param[in] func функция, которая исполнится, если будет получен элемент очереди.
    /// @return true, если элемент был обработан, иначе false.
    bool process( const std::function< void( const T& ) >& func );

private:
    /// @brief Узел односвязного списка.
    struct Node
    {
        T data;
        std::unique_ptr< Node > next;
    };

    const Node* get_back_locked();

    std::condition_variable cond_;          ///< условие для ожидания.
    std::mutex front_mutex_;                ///< мьютекс для блокировок доступа к первому элементу.
    std::mutex back_mutex_;                 ///< мьютекс для блокировок доступа к последнему элементу.
    std::unique_ptr< Node > front_;         ///< начало очереди.
    Node *back_;                            ///< конец очереди.
    bool finish_;                           ///< условие окончания работы.
};
```

Очередь основана на односвязном списке. Отличие от предыдущего класса заключается в том, что блокировки разных концов очереди происходят отдельно. Есть два мьютекса: на передний и задний концы очереди. Один блокируется при добавлении элементов, а другой – при удалении. Чтобы проверить, что в очереди не один элемент, на время проверки блокируются оба мьютекса. В очереди всегда есть пустой элемент для отделения начала от конца.

3. Сравнение потокобезопасных очередей с блокировками.

Было проведено сравнение очередей. При измерениях очередь обрабатывала 600 задач по умножению матриц 10x10. Во всех случаях бралось среднее значение времени работы из 1000 запусков программы.

В табл. 1 показано время работы обеих очередей при 5 потребителях и 5 производителях.

Таблица 1. Сравнение очередей при 5 потребителях и 5 производителях.

Очередь	User time, мс	System time, мс	Real time, мс
«Грубые» блокировки	34.081	9.264	27.745
«Тонкие» блокировки	34.040	9.314	28.254

В табл. 2 приведено сравнение очередей при 8 потребителях и 2 производителях.

Таблица 2. Сравнение очередей при 8 потребителях и 2 производителях.

Очередь	User time, мс	System time, мс	Real time, мс
«Грубые» блокировки	31.782	8.827	27.232
«Тонкие» блокировки	31.572	7.854	28.255

В табл. 3 приведено сравнение очередей при 2 потребителях и 8 производителях.

Таблица 3. Сравнение очередей при 2 потребителях и 8 производителях.

Очередь	User time, мс	System time, мс	Real time, мс
«Грубые» блокировки	33.266	10.973	27.310
«Тонкие» блокировки	33.069	10.305	27.460

Видно, что при использовании «тонких» блокировок время ожидания в блокировках (system time) вплоть до 10% меньше, чем при «грубых» блокировках, особенно, когда потоков-потребителей и производителей не равное количество. Но полное время работы программы (real time) всё равно больше при «тонких» блокировках. Так происходит из-за дополнительных расходов на выделение динамической памяти.

Выводы.

В работе были исследованы потокобезопасные очереди с блокировками. Было использовано два вида блокировок: «грубые» и «тонкие».

Было установлено, что «тонкие» блокировки позволяют добиться меньшего времени ожидания в блокировках, чем «грубые», но не дают ускорения из-за накладных расходов при их использовании. Время ожидания в «тонких» блокировках меньше всего, когда потоков-производителей и потоков-потребителей неравное количество и большинство потоков простаивает. При «грубых» блокировках подобное простаивание длится дольше, потому что все потоки блокируются на одном мьютексе, а при «тонких» блокировках – на двух.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Исходный код программы доступен в репозитории https://github.com/Astana-Mirza/parallel_algo/tree/master.