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 структур данных без блокировок

Студентка гр. 0304 _____ Говорющенко А.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучить принцип построения потокобезопасных структур данных без блокировками.

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.

В отчёте: Сравнить производительность с реализациями структур данных из работы В отчёте сформулировать инвариант структуры данных.

Выполнение работы.

Для выполнения лабораторной работы была написана lock-free очередь Майкла-Скотта, удовлетворяющая гарантии прогресса. Для очереди был написан класс LockFreeQueue, а для безопасной работы с освобождением памяти использована схема указателей опасности (hazard pointers).

В основе реализации очереди лежит односвязный список, имеющий два публичных метода: добавление в конец (push) и удаление с начала (pop). Оба метода содержат бесконечные циклы, в которых происходят попытки атомарной замены указателей с помощью операции CAS, которая в C++ реализована через функции compare_exchange_weak и compare_exchange_strong. В случае, если сторонний поток помешал текущему потоку выполнить действие, то цикл повторится снова, и так до тех пор, пока операция удаления/вставки не будет выполнена.

Для того, чтобы алгоритм удовлетворял гарантии lock-free прогресса, необходимо заканчивать операции за другими потоками, если будет обнаружено промежуточное состояние очереди.

Такая ситуация может случиться когда во время добавления элемента в очередь локальный указатель на конец очереди будет отличаться от глобального и не будет концом вовсе. Тогда, вместо ожидания завершения операции другим потоком, текущий поток самостоятельно, с помощью операции CAS, передвинет указатель с локального конца очереди на элемент, следующий за ним. Таким образом, в случае, если другие процессы будут приостановлены, текущий поток сможет продолжить свою работу, не ожидая завершения других.

Аналогично происходит и с извлечением элемента из очереди и указателем на голову. Если было замечено, что указатель, который был взят, уже не является актуальным указателем на голову, и того указателя уже нет, то в цикле while операция повторяется с верным значением указателя на начало.

Беря во внимание ранее озвученные факты, заметим, что указатели на начало и конец могут не совпадать с фактическим началом и концом. Поэтому инвариант структуры данных без блокировки можно сформулировать так: указатель на начало очереди всегда указывает на элемент, стоящий не после элемента под указателем на конец. При этом они могут указывать на один и тот же, например, в ситуации, когда очередь пуста и добавление элемента еще не завершилось.

Безопасное освобождение памяти

При удалении элементов сразу из очереди, может возникнуть ситуация, когда один поток еще использует ту часть памяти, которую другой поток освобождает, и разыменовываясь по своей локальной копии указателя обратиться к несуществующему элементу.

Чтобы избежать подобной ситуации, был выбран один из алгоритмов безопасного освобождения памяти — hazard pointers, так как является одним из наиболее известных и проработанной схемой отложенного удаления. Эта схема основана только на атомарных чтении и записи и не использует такие примитивы синхронизации, как CAS.

Суть схемы заключается в том, что для каждого потока выделяется H ячеек под указатели опасности, то есть те указатели, которыми в данный момент используются в операциях. Данное число выбирается в зависимости от структуры данных, для очереди достаточно H = 2, один для операции добавления, и два для операции извлечения. Число потоков T должно быть известно заранее.

В каждом потоке также храниться массив указателей для отложенного удаления. При извлечении элементов из очереди потоки добавляют извлеченные элементы в данные массивы, и в момент, когда массив будет заполнен и его размер будет равен R (R сравнимо с N = H*T, но более N; например, R = 2N), будет вызвана процедура освобождения памяти scan(). Условие R > H*T является существенным: только при выполнении этого условия гарантируется, что scan() сможет что-нибудь удалить из массива отложенных данных. Если это условие нарушено, scan() может ничего не удалить из массива, и мы получим ошибку алгоритма – массив полностью заполнен, но уменьшить его размер невозможно.

Сравнение реализаций многопоточной очереди

В таблицах 1-3 приведен сравнительный срез времени работы многопоточных очередей с «грубыми», «тонкими» блокировками и очередей без блокировок. Замеры производились на количестве задач 300 с матрицами размера 10x10 при запуске программы 100 раз.

Таблица 1. Сравнение очередей при 10 производителях и 10 потребителях.

Очередь	User time, мс	System time, мс	Real time, мс
«Грубые»	22.099	15.295	25.891
блокировки «Тонкие»	21.716	18.071	25.089
блокировки Lock-free	33.996	12.862	27.097

Таблица 2. Сравнение очередей при 4 производителях и 10 потребителях.

Очередь	User time, мс	System time, мс	Real time, мс
«Грубые»	9.474	5.497	10.247
блокировки «Тонкие»	9.735	5.097	10.358
блокировки Lock-free	20.188	3.681	10.905

Таблица 3. Сравнение очередей при 10 производителях и 4 потребителях.

Очередь	User time, мс	System time, мс	Real time, мс
«Грубые»	21.740	18.613	25.370
блокировки «Тонкие»	22.370	21.343	25.616
блокировки Lock-free	25.618	15.493	25.848

По результатам проведенных измерений, видно, что операции с очередью lock-free занимают больше времени, чем операции с блокировками,

особенно разница заметна, когда производителей меньше, чем потребителей, и потребители простаивают в цикле без блокировок. User time в такой ситуации превышает время аналогов с блокировками в 2 раза.

Sys time во всех случаях у очереди без блокировок немного меньше времени версий с блокировками, а real time в среднем превышает на несколько единиц.

Одной из причин уменьшения производительности является частое выделение и освобождение динамической памяти. В lock-free очереди выделение памяти происходит больше, чем в очередях с блокировками, так как выделяется память не только для элементов очереди, но и для работы схемы указателей опасности.

Выводы.

В ходе работы были изучены принципы работы с потокобезопасной очередью без блокировок. Был сформирован инвариант lock-free очереди: указатель на начало очереди всегда указывает на элемент, стоящий не после элемента под указателем на конец. Также было выявлено, что производительность lock-free очереди хуже, чем у очередей с блокировками, так как ее реализация требует большего выделения и освобождения памяти.