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 без блокировок

Студентка гр. 1304 _____ Чернякова В.А.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2024

Цель работы.

Изучить принцип построения потокобезопасных структур данных без блокировок.

Задание.

Реализовать очередь, удовлетворяющую lock-free гарантии прогресса (очередь Майкла-Скотта). Протестировать доступ к реализованной структуре данных для случаев разного количества потоков производителей и потребителей (аналогично работе 2).

Дополнительно (выполнение работы на отл.): реализовать корректное освобождение памяти.

В отчёте: сравнить производительность с реализациями структур данных из работы 2.

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.

Сравнение потокобезопасных очередей с блокировками.

Сравнение осуществлялось при помощи измерений обработки очереди 300 задач по умножению матриц 100x100.

Производители	Потребители	Блокировка	Real Time, сек	Sys Time, сек
2	2	Грубая	5.384	0.189
		Тонкая	5.541	0.101
		lock-free	19.434	2.526
12	12	Грубая	2.661	0.478
		Тонкая	2.678	0.388
		lock-free	7.700	19.261
16	1	Грубая	10.089	0.732
		Тонкая	10.491	0.546
		lock-free	40.864	49.884
1	16	Грубая	2.565	0.159
		Тонкая	2.573	0.101
		lock-free	5.965s	1.180
500	500	Грубая	3.788	9.666
		Тонкая	3.630	8.423
		lock-free	24.080	50.074
500	10	Грубая	3.221	9.686
		Тонкая	3.076	7.859
		lock-free	9.883	47.417
10	500	Грубая	2.845	1.278
		Тонкая	2.997	0.981
		lock-free	26.033	1.800

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.

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.

Lock-free алгоритмы часто сложнее и требуют больше операций для предотвращения состояния гонки, чем простые блокировки. Они используют атомарные операции, такие как compare-and-swap (CAS), которые могут быть дорогими по времени исполнения из-за постоянных проверок и повторных

попыток. При высокой конкуренции за ресурсы многопоточные процессы могут часто сталкиваться с неудачными попытками обновления, увеличивая общее время исполнения. В то же время грубая блокировка (coarse-grained lock) ограничивает доступ целиком, что может быть быстрее в ситуациях с меньшей конкуренцией, но менее масштабируемо. Тонкая блокировка (fine-grained lock) может сократить время ожидания за счёт частичного разделения ресурса, но при этом она проста в реализации и снижает нагрузку по сравнению с lock-free подходом.

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.