Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт естественных наук и математики
Кафедра алгебры и дискретной математики

Исследование потокобезопасных неблокирующих структур данных

Допустить к защите:
Зав. кафедрой
доктор физико-математических наук,
профессор Волков М. В.

Выпускная квалификационная работа
студента 4 курса
Сваловой А. А.

Научный руководитель:
Плинер Ю. А.

Екатеринбург
2017 год

Содержание

1 Введение

С ростом прогресса многие электронные устройства становятся многоядерными, появляются многопроцессорные устройства. Поэтому задача программиста, как человека, который пытается максимально хорошо использовать предоставленные ресурсы, - писать программы, способные масштабироваться и параллелиться. Поэтому сейчас все чаще и чаще пишут многопоточные программы и используют многопоточные структуры данных. Такие программы могут выполнять сразу несколько инструкций на каждом процессоре или ядре, однако такой код обладает рядом проблем, связанных с доступом к общим ресурсам разными потоками или процессами.

Если два или более потока захотят изменить один и тот же участок памяти, то они попытаются сделать это единовременно. После выполнения операции неизвестно, как будет выглядеть этот участок памяти, так как порядок выполнения инструкций разных потоков неопределен. Возникает вопрос: как в таком случае контролировать доступ к этому ресурсу? Хочется чтобы в каждый момент времени, способом, очевидным для разработчика, ресурсом владел только один поток, а все остальные каким-то образом ждали своей очереди. Такое поведение можно осуществить несколькими способами.

Самый простой из них - блокировка. Каждый раз, когда поток хочет сделать что-то с ресурсом, он проверяет, нет ли блокировки на этот ресурс. Если есть, поток ждет, пока блокировка не освободится, если нет, то он пытается первым захватить блокировку. В случае успешного захвата он осуществляет все операции с ресурсом и освобождает блокировку. В это время все остальные потоки ждут этот и ничего не делают.

Такой механизм синхронизации очень прост в понимании и реализации,

учитывая существование встроенных блокировок в большинство современных ОС. Также этот способ, очевидно, позволяет только одному потоку единовременно получить доступ к ресурсу. Однако, в данном способе существует и масса проблем, которые сводят на “нет” все преимущества. Во-первых, при большом количестве потоков, желающих получить доступ к ресурсу, возникает “узкое горлышко”, т. е. место в программе, которое тормозит выполнение программы в целом. Во-вторых, при существовании больших участков программы с блокировкой теряется весь смысл многопоточности. В эти участки все равно может заходить только один поток, как и в однопоточном программировании. В-третьих, существуют некоторые особенности операционной системы: переключение потоков - дорогая операция. При долгом ожидании освобождения ресурса происходит очень большое количество переключений, следовательно, большое количество времени тратится на бесполезные операции. В-четвертых, возможны ситуации, когда один поток захватил первый ресурс и ждет освобождение второго ресурса, в то время как второй поток захватил второй ресурс и ждет освобождения первого. Такая ситуация называется взаимная блокировка (deadlock). Программа в таком случае останавливает свое выполнение совсем и не может без каких-либо вмешательств извне разрешить эту ситуацию.

Эти проблемы привели исследователей к созданию других способов синхронизации. Один из них - неблокирующая синхронизация. Это способ, при котором каждый поток пытается применить низкоуровневые атомарные аппаратные примитивы, а не использовать блокировки. Таким образом в каждый момент времени выполняется только одна операция, только одного потока. Все остальные операции в других потоках либо завершаются ошибкой, либо

выполняются сразу следом за предыдущей. Такие алгоритмы обеспечивают общее продвижение программы в целом: даже если какой-то поток не смог выполнить операцию или завершился с ошибкой - значит, что какой-то другой поток успешно выполнил свою операцию. Не существует случаев, когда все потоки одновременно простаивают, и как частный случай этого, невозможно существование взаимных блокировок.

Однако, несмотря на все преимущества, данная область является до сих пор развивающейся. Нельзя просто взять и написать неблокирующую реализацию алгоритма, основанного на блокировках. В некоторых случаях это оказывается легко, в некоторых до сих пор не придумано неблокирующих аналогов. Причина: каждый раз нужно творчество, чтобы свести все операции над разделяемым ресурсом к последовательности независимых атомарных операций, т. е. не существует универсального способа написания неблокирующей реализаци. Однако сложность реализации и изобретения алгоритма часто стоит усилий. Пусть этот класс алгоритмов совсем не о скорости работы, а о гарантии продвижения системы в целом, но в итоге большинство неблокирующих алгоритмов имеют в среднем ожидаемую сложность меньше, чем блокирующие аналоги. Но это только в теории. На практике скорость работы зависит от конкретной реализации, области применения, часто встречающихся запросов и т. д.

Цель данной работы: реализовать основные структуры данных, реализующие интерфейс ISet и на практике выявить являются ли неблокирующие алгоритмы эффективней блокирующих, какие алгоритмы вообще реально применимы, и выяснить, как адаптировать алгоритмы, разработанные под языки программирования с неуправляемой памятью (понять, как по-русски

managed), к языкам с управляемой памятью.

В работе представлены структуры данных, реализующие интерфейс ISet. Данный интерфейс включает в себя добавление элемента в множество, удаление элемента, а также поиск и перечисление всех элементов в множестве. Этих сценариев достаточно, чтоб понять, как ведут себя различные реализации на практике. Для сравнения были выбраны следующие реализации: сортирующийся лист, хэш-таблица, скип-лист и дерево поиска. Также взяты готовые реализации всех этих структур из библиотеки языка C#, чтобы сравнить неблокирующие реализации с блокирующими. В приложении приведены различные результаты сравнений всех этих структур и вариации использования их в реальной жизни. Все алгоритмы адаптированы под язык C# и собраны в один общий модуль с внешним интерфейсом ISet.

2 Глава 1. Основные определения

2.1 Атомарная операция

Все неблокирующие алгоритмы можно разделить на три типа: Waitfree, Lockfree, Obstruction-free.

В первом типе каждый поток совершает каждую операцию за конечное число шагов, независимо от влияния других потоков. это самое сильное требование из-за чего редко реализуемое. Такие алгоритмы обычно реализуют атомарный инкремент или атомарную замену ссылок.

Во втором типе система в целом двигается вперед, даже если какой-то поток стоит на месте. Если какой-то поток не смог выполнить операцию, значит, что какой-то другой поток смог выполнить свою операцию, следовательно, в целом система продвинулась. Эти алгоритмы обычно реализуют атомарное сравнение и замену.

В третьем типе каждый может выполнить каждую операцию за конечное количество шагов, если ничего ему не помешает. В данном случае может случиться ситуация, когда ни один из потоков не двигается вперед, однако ни один заблокированный поток не может мешать работе всех остальных потоков, следовательно, это все равно более сильная гарантия, чем блокирущие алгоритмы.

Каждая из этих реализаций использует абстракцию “атомарная операция” - это операция, которая либо не выполняется совсем, либо выполняется как единое целое. В данной работе используется атомарная операция Compare And Swap (CAS). Эта операция сравнивает две ссылки и, если они равны, меняет первую на данную третью. Эта операция предоставляется боль-

шинством операционных систем и уже встроена в язык C#.

2.2 Atomic markable reference(придумать русское название)

Некоторые алгоритмы с неблокирующей синхронизацией используют

2.3 SpinWait

3 Глава 2. Реализации алгоритмов

4 Глава 3. Тестирование

5 Заключение

6 Список литературы

7 Приложения