**1)Вступление**

Здравствуйте, уважаемый председатель и члены государственной экзаменационной комиссии. Вашему вниманию представляется выпускная квалификационная работа магистра на тему Анализ потокобезопасных структур данных.

**2) Цели, задачи и актуальность**

Анализ потокобезопасных структур данных – это одна из самых важных тем для разработчиков ПО. Затраты на анализ и разработку потокобезопасных структур данных сильно замедляют процесс создания программных продуктов.

Сегодня я хочу рассказать вам о потокобезопасных структурах данных в параллельном программировании, их преимуществах и проблемах. Любой анализ должен состоять из определенного списка шагов. В контексте анализа потокобезопасных структур данных решались следующие задачи:

1. Изучение основных достоинств и недостатков параллелизма.

2. Определение критериев анализа.

3. Выбор структур данных для анализа.

4. Создание тестовых случаев.

5. Проведение экспериментов.

6. Анализ результатов.

Цель работы состоит в изучении слабых мест потокобезопасных структур данных и выявлении методов оптимизации с точки зрения скорости работы.

Задачи

1. Изучение различных структур данных и параллелизма.

2. Изучение общих методов анализа потокобезопасных структур

данных.

3. Сравнительный анализ производительности потокобезопасных и

непотокобезопасных структур данных.

4. Реализация различных lock-free очередей.

5. Тестирование lock-free очередей.

**3)Разделяемые структуры данных и потокобезопасность**

**“Текст со слайда”**

**4)Конкурентность**

**“Текст со слайда”**

**5)Преимущества параллелизма**

Давайте начнем с преимуществ параллелизма. Параллельные вычисления могут повысить производительность программ и ускорить выполнение задач за счет возможности выполнять несколько задач или операций одновременно.

**“Текст со слайда”**

**6)Проблемы параллелизма**

Однако, параллельные вычисления имеют свои проблемы. Например, может возникать такое частое явление, как состояние гонки(race condition), когда работа системы или приложения зависит от того, в каком порядке выполняются части кода.

**“Текст со слайда”**

**7)Изучение общих методов анализа потокобезопасных структур данных**

Чтобы избежать этих проблем, мы должны изучить общие методы анализа потокобезопасных структур данных. Мы должны применять эти методы для разработки структур данных, которые безопасны для использования в многопоточных приложениях.

**“Текст со слайда”**

Статический анализ — это метод анализа программного кода без его фактического выполнения. Статический анализатор может проверять код на наличие ошибок, предупреждать о потенциальных проблемах, определять уязвимости и прочее.

В контексте потокобезопасных структур данных, статический анализ может помочь выявить возможные проблемы с параллельным доступом к данным, такие как условия гонки, гонки на запись и прочие ошибки синхронизации. Он также может помочь выявить потенциально небезопасные конструкции в коде, которые могут привести к конфликтам при параллельном доступе к данным.

Формальная верификация — это метод формального доказательства корректности программного обеспечения. Она используется для проверки соответствия программного кода спецификации, которая формулирует требования к программе. Формальная верификация может помочь гарантировать отсутствие ошибок, таких как условия гонки, при работе с потокобезопасными структурами данных.

Динамический анализ для потокобезопасных структур данных — это метод анализа, который основывается на наблюдении за выполнением программы во время ее выполнения. Он позволяет выявлять проблемы с потокобезопасностью в реальном времени, что может быть особенно полезным для обнаружения проблем, которые могут произойти только при определенных условиях во время выполнения программы.

Один из подходов к динамическому анализу потокобезопасных структур данных — это использование инструментов для отслеживания состояния памяти во время выполнения программы. Эти инструменты могут использоваться для обнаружения условий гонки, когда два или более потока пытаются одновременно получить доступ к общим данным. Они также могут использоваться для обнаружения других проблем, таких как взаимоблокировки и неправильное использование примитивов синхронизации.

**8)Сравнительный анализ производительности потокобезопасных и непотокобезопасных структур данных**

Для того, чтобы определить, какие структуры данных наилучшим образом подходят для параллельных вычислений, необходимо провести сравнительный анализ производительности потокобезопасных и непотокобезопасных структур данных. В этом анализе мы можем определить, какие структуры данных являются наиболее эффективными для выполнения определенных задач в многопоточных приложениях.

**“Текст со слайдов”**

1. Затраты на синхронизацию. Потокобезопасные структуры данных требуют дополнительных затрат на синхронизацию доступа к данным из разных потоков, что может приводить к снижению производительности. В то же время, непотокобезопасные структуры данных не требуют таких затрат и могут работать быстрее в однопоточных приложениях.

2. Конкуренция за ресурсы. В многопоточных приложениях потокобезопасные структуры данных могут снижать конкуренцию за ресурсы, так как позволяют нескольким потокам одновременно обращаться к данным без блокировки. В то же время, непотокобезопасные структуры данных могут приводить к конкуренции за ресурсы, что может привести к блокировке и снижению производительности.

3. Параллелизм. Потокобезопасные структуры данных позволяют эффективно использовать параллелизм и ускорить выполнение операций в многопоточных приложениях. В то же время, непотокобезопасные структуры данных могут приводить к некорректным результатам при параллельном выполнении операций.

4. Размер данных. Потокобезопасные структуры данных могут иметь больший размер и большую сложность, чем непотокобезопасные структуры данных, что может приводить к снижению производительности при работе с большими объемами данных. Однако, это зависит от конкретной структуры данных и ее реализации.

5. Ошибки и безопасность. Потокобезопасные структуры данных предотвращают ошибки, которые могут возникать при работе с данными из нескольких потоков. Также они обеспечивают безопасность данных при работе в многопоточных приложениях. В то же время, непотокобезопасные структуры данных могут нарушать целостность данных при работе в многопоточной среде, что может привести к некорректным результатам и ошибкам.

**9)Реализация различных lock-free очередей**

Для реализации потокобезопасных структур данных было принято использовать lock-free очереди, так как они являются более сложными, чем lock-free стэки. Lock-free очереди являются структурами данных, которые могут работать без блокировок, что повышает их отказоустойчивость. Были выбраны алгоритмы MSQueue, Optimistic Queue, Basket Queue. Optimistic Queue и Basket Queue являются потомками и в некоторой степени модификациями MSQueue. Для все этих алгоритмов, точкой линеаризации является CAS. CAS или же Сompare And Swap это атомарная инструкция, используемая в многопоточности для достижения синхронизации. Она сравнивает содержимое ячейки памяти с заданным значением и, только если они совпадают, изменяет содержимое этой ячейки памяти на новое заданное значение. Это выполняется как одна атомарная операция.

**10)MSQueue**

Основная идея Michael-Scott Queue заключается в использовании двух указателей (head и tail), которые указывают на начало и конец очереди соответственно. Новые элементы добавляются в хвост очереди, а порядок изъятия элементов определяется порядком, в котором они были добавлены.

Michael-Scott Queue реализована таким образом, что оба указателя гарантированно обновляются с использованием CAS-операций, что позволяет множеству потоков безопасно работать с очередью.

Эта структура данных хорошо работает в режиме producer-consumer, когда producer могут эффективно добавлять элементы в хвост очереди, а consumer может извлекать элементы с любого места очереди. Michael-Scott Queue является эффективной структурой данных в ситуациях, когда многие потоки работают с одной очередью и исполнение потоков равномерно распределено между производителями и потребителями.

Нельзя выполнить добавление элемента в очередь и перемещение T атомарно. В таком случае, стальные потоки помогают перенести указатель на хвост очереди. Если поток видит непустой T.next (то есть если он провалил CAS(tail.next,null,newTail)), то он должен помочь перенести T, то есть выполнить CAS(T,tail,tail.next.get()) однократно. Если CAS выполнен успешно, то хвост перемещён успешно (а значит, наш поток должен вернуться к добавлению нового элемента). Если же он выполнен неудачно, то это значит, что T уже не указывает на tail, а значит, другой поток уже успешно переместил хвост (а значит, наш поток должен вернуться к добавлению нового элемента). Аналогично, замыкающий поток должен перенести указатель в случае с pop.

**11)Optimistic Queue**

Она использует оптимистический метод вставки элементов, который не требует блокировок в большинстве случаев, что обеспечивает высокую производительность и отказоустойчивость.

Под капотом используется самовосстанавливающееся решение. В рамках их алгоритма прямая связь, то есть прямое направление от головы к хвосту, будет всегда существовать и являться актуальной и корректной. Обратная же связь, то есть направление от хвоста к голове, будет являться динамической и может иметь нарушения в порядке. Однако, при обнаружении таких нарушений в порядке, мы всегда сможем их устранить. При обнаружении нарушения, алгоритм просто пройдет по next ссылкам, и на их основании восстановит порядок prev ссылок. Фактором нарушения является невыполнение неравенства pHead->prev->next != pHead.

**12)Basket Queue**

Она представляет собой очередь в виде набора логических корзин (basket), каждая из которых доступна для добавления нового элемента некоторый краткий интервал времени. Интервал прошел — создается новая корзина.

Каждая корзина — это неупорядоченный набор элементов. Казалось бы, при таком определении нарушается основное свойство очереди — FIFO, то есть очередь становится не совсем очередью (unfair). FIFO соблюдается для корзин, но не для элементов в корзинах. Если интервал доступности корзины для добавления достаточно мал, можно пренебречь неупорядоченностью элементов в ней.

Как определить длительность этого интервала? Рассмотрим очередь MSQueue. В операции enqueue при высокой конкуренции, когда CAS изменения хвоста не сработал, то есть там, где в MSQueue вызывается back-off, мы не можем определить, в каком порядке элементы будут добавлены в очередь, так как они добавляются одновременно. Это и есть логическая корзина. По сути получается, что абстракция логических корзин — это разновидность back-off стратегии.

**13)Basket Queue**

1. Потоки A, B и C хотят добавить элементы в очередь. Они замечают, что хвост находится на своем месте (как известно, в MSQueue хвост может указывать на любой элемент, а не на последний). Таким образом, они пытаются изменить его одновременно.
2. Поток A побеждает и успешно добавляет элемент. В то время как потоки B и C оказываются неудачниками - их CAS с хвостом проваливается. В результате они начинают добавлять свои элементы в корзину, используя старое значение хвоста, которое было прочитано ранее.
3. Поток B быстрее всех и успешно добавляет свой элемент. А в это же время поток D также вызывает enqueue и добавляет свой элемент, изменяя хвост.
4. Поток C тоже завершает добавление, но в середину очереди. Он использует старый указатель на хвост, который он прочитал при первоначальном входе в операцию, перед неудачной попыткой выполнения CAS.

**14)Тестирование lock-free очередей**

Для тестирования работоспособности наших структур данных было принято решение провести два основных теста.

Первый тест - Producer/Consumer test, позволяет определить производительность структуры данных и то, насколько эффективной является реализация алгоритма в реалистичном сценарии для очереди, например, когда происходит всплеск активности. Очередь предназначена не для безумного накопления, а для буферизации всплесков активности, и нормальное состояние очереди — это отсутствие в ней элементов.

Второй тест - Endurance test, позволяет определить, насколько эффективно будет работать структура данных под нагрузкой, а также оценить влияние аллокатора на произодительность.

**15)Producer/Consumer test**

**“Описать подробно графики”**

**16)Endurance test**

**“Описать подробно графики”**

**17)Заключение**

В заключении, параллельные вычисления в программировании — это инструмент, который может повысить производительность программы и обеспечить отказоустойчивость системы. Однако, для того чтобы использовать этот инструмент безопасно и эффективно, необходимо анализировать сферу применения, и применять корректные реализации потокобезопасных структур данных.

В результате выполнения выпускной квалификационной работы был проведен анализ потокобезопасных структур данных. В частности, были реализованы две программы. Первая содержит внутри себя реализации различных алгоритмов lock-free очередей, а также тесты. Вторая используется для отображения и анализы графиков производительности.

Благодаря проведенном анализу, компании и команды снизят затраты на разработку и реализацию параллельных приложений. Это произойдет за счет увеличения скорости разработки, так как не потребуется подробного изучения и анализа потокобезопасных структур данных.