Автор: Ольга Черникова
Лектор: Калишенко Евгений
Курс по параллельному программированию
СпбАУ, 2017 год, 3 курс, 6 семестр, 302 группа

Lock-free и wait-free алгоритмы.

Проектирование Lock-free алгоритмов.

Постановка задачи.

У нас есть два атомарных регистра. Причем эти регистры - SWMR(писать может один **выделенный** поток, читать - несколько). Хотя для lock-free это будет не очень важно.

Названия регистров:	v1	v2

Хочется спроектировать алгоритм, который будет делать снепшот этих двух регистров. Причем снепшот должен обладать двумя свойствами.

1. ***Консистентность*** - гарантия, что был момент времени, когда было такое состояние регистров.
2. ***Актуальность*** - этот момент времени с данным состоянием регистров был после того, как мы начали делать снепшот.

Приведем пример не консистентного снепшота. Метод который снимает снепшот назовем **scan.**

Название регистров:	v1	v2
Поток, который считывает регистры считал из первого регистра 0, из второго еще ничего не считал.	**0**	0
Первый поток поменял значение.	1	0
Поменяли значение второго регистра и скан считал со второго регистра 2. В результате вернул значения (0, 2), а такого состояния ни разу не было.	1	**2**

Если не будет выполнятся актуальность, то можно всегда возвращать 0, то есть то, что было в регистрах с самого начала. Нам нужно все-таки что-то более новое возвращать.

Автор: Ольга Черникова
Лектор: Калишенко Евгений
Курс по параллельному программированию
СпбАУ, 2017 год, 3 курс, 6 семестр, 302 группа

Что можно хранить в регистрах.

Атомарные регистры в полном смысле этого слова. То есть можно пользоваться всеми хаками cas. В cas мы можем передать только что-то не больше машинного слова. А если в нашем регистре мы хотим хранить нечто большее нашего машинного слова, то тогда мы используем указатели. То есть создаем структуру, в нее кладем информации столько много сколько хотим и пользуемся дальше уже указателем на структуру, который уже нормально занимает машинное слово. Тогда атомарность нормально будет выполняться.

Но одно важное свойство должно быть у этой структуры, а именно она должна быть ***immutable***. То есть при любом изменение структуры, структура сначала копируется.

В атомарные регистры можно записывать все что угодно.

Lock-free алгоритм.

Идея. Добавляем к каждому регистру еще и счетчик. То есть номер версии. Когда меняем значение регистра, инкрементируем значение счетчика. Когда нужно будет взять снепшот, снимаем вместе с номерами версий, а потом пробегаемся и проверяем, что версии все еще те же.

Регистры, версии	v1, T1	v2, T2
Считали (0, 0) -	(0, 0)	(0, 0)
	(0, 0)	(13, 1)
	(5, 1)	(13, 1)
Считали (0, 0) (6, 2)	(5, 1)	(6,2)

Еще раз пробежались, проверили, что что-то не совпадает, считали еще раз все заново.

Теперь вопрос, почему нельзя было сделать все тоже самое, когда у нас не было номера версии. То есть считывать до тех пор, пока значения не совпадут. Сейчас приведем пример.

Значение регистров:	v1	v2
Считываем из первого регистра. Получаем: 0 -	0	0

	1	0
Считываем из второго регистра. Получаем: 0 2	1	2
	1	3
Проверяем первый регистр. **0** 2	0	3
	1	3
Проверяем второй регистр. **0 2**	1	2

Получаем снепшот 0, 2, хотя такого никогда не было.

Проектирование wait-free алгоритмов.

Нужно сделать тоже самое, только нужно гарантировать, что за определенное количество шагов каждый поток решит свою задачу и не будет бесконечного ожидания, ни в каком случае. И это будет лучше чем Lock-free. Точнее, это будет lock-free и при этом мы даем еще гарантий.

Честный Lock-free

Есть честные lock-free, есть получестные. Такие, которые чуть-чуть spin lock используют. С теоретической точки зрения они не lock free, но с теоретической они бывают быстрее. На некоторых данных, и используют они на правильных. Есть четный, а есть абсолютно честный. Абсолютно честный не использует никаких блокирующих системных вызовов. Самый часто потоко безопасный системный вызов это **new.** Он потоко безопасный и блокирующий. Есть оперативная память, есть ядро. Для ядра операционной системы оперативная память это ресурс и для этого ресурса есть примитив синхронизации. И когда вызывается new, примитив синхронизации лочится. При вызове new из разных процессов, они лочатся на одном примитиве синхронизации. И если какое-то приложение много работает с памятью, то это влияет на другие приложения. На самом деле будут использовать некоторые оптимизации.

То, что мы придумывали со списком, был не честный lock-free. Если бы поток внутри ядра остановился бы навсегда, тогда бы мы уже больше бы никакую бы память не выделили и

не закончили бы свою работу и никто бы не прогрессировал. Тут тонкая разница между честным и абсолютно честным.

Размышление по поводу wait-free.

Суть алгоритма в следующем. В каждом регистре будем хранить весь снепшот всех регистров. Как он будет туда попадать. Пусть мы умеем снимать снепшот. Версии мы так же храним. Есть функция scan, которая wait-free.

```
Update () {
        Snapshot s = scan();
        write(v, T++, s); // записали новое значение, версию и снепшот без последнего изменения.
}
```

Теперь метод снятия снепшота. Для того, чтобы снять снепшот, нам достаточно наблюдения изменения регистра не более двух раз.

- Считываем регистры в первый раз.
- Считываем второй раз оба регистра
    - Обе версии оказались одинаковые. Тогда можем отдать снепшот и отдать как в lock-free
    - Версии не равны. Например версии в регистре 1 не равны. Это значит между первым и вторым чтением произошла запись в первый регистр. Вместе с этой записью создался снепшот. Можно ли вернуть снепшот из первого регистра, видя, что вот он только что поменялся? Нет, нельзя. Потому что он мог быть неактуальным. Снепшот в регистре 1 мог начать сниматься до того, как мы начали функцию scan. То есть он не актуальный.
        - Читаем третий раз. Может быть две ситуации.
            1. Вторые и третья версии равны. Можем тогда эту версию вернуть.
            2. Пусть не равны для того же самого регистра v1. Тогда снятие снепшота могло начаться только после первйо записи в v1. Потому что только один поток мог писать в один регистр. Тогда можем вернуть снепшот из первого регистра. Победа.
            3. После третьего чтения на этот раз изменился второй регистр, тогда мы опять ничего не можем вернуть. Снепшот для второго регистра мог начаться очень рано.
                - Считываем еще раз и аналогично предыдущему случую, либо все хорошо и возвращаем или кто-то не совпал из того кто не совпал возвращаем снепшот.

Автор: Ольга Черникова
Лектор: Калишенко Евгений
Курс по параллельному программированию
СпбАУ, 2017 год, 3 курс, 6 семестр, 302 группа

Теперь сколько потребуется чтений, если у нас n регистров. Тогда нам потребуется n + 2 чтения всех регистров.