Параллельные и распределенные вычисления. Задание 1 “Читатели-писатели”.

Старченко Владимир

Основная часть

Обоснование решения

Необходимо решить довольно распространенную проблему: есть алгоритм с блокирующими операциями чтения и записи, требуется сделать блокирующюю запись, но не блокирующее чтение. Было предложено использовать мьютексы и условные переменные, чтобы переключаться между операций записи и чтения. Пока идет запись, никто больше ничего не может сделать. Пока идет чтение, мы разрешаем чтение из других потоков и запрещаем запись.

Описание реализации

Есть несколько разных подходов реализации, которые направлены на решение разных задач. Например, в случае когда при приходе запроса на обновление данных, прочитанные данные сразу станут невалидными, запись будет иметь больший приоритет. И наоборот, может так случиться, что нам нужно стабильно выдавать значение пользователю, может немного устаревшие, но быстро и без перебоев.

Конкретно в данном случае у нас проверялись тесты на голодающих писателей и читателей. Это значит, что нам требуется, чтобы и писатели, и читатели не стояли в очереди слишком долго и не пропускали других вперед всё время.

В моей реализации делается следующим образом.

Если есть писатель, даем ему писать. Новый, пришедший писатель становится ждущим. В этот момент остальные читатели спят на конд варе. Читатели ждут пока активный писатель отпустит мьютекс. Перед окончанием работы писателя будим всех читателей, они становятся активными.

Читатели становятся активными, пересчитываются в переменную “active_readers” и начинают работать параллельно. Писатели при этом не могут начать читать так как спят на конд варе. Когда читатель заканчивает работу, он уменьшает счетчик “active_readers”. Последний читатель (тот что обнулил счетчик) будит писателя. Если есть ожидающий писатель, новые читатели ждут на конд варе. Когда последний читатель заканчивает, он будит писателя.

Если есть ждущий писатель (а значит есть и работающий), читатели не могут стать активными (остаются ждущими), кроме как разбужены писателем, закончившим работу.

Такой механизм позволяет чередовать писателей и читателей, избегая состояния голодания. И по возможности (если никакой поток на подвисает на уровне системы на бесконечно долго и всегда пропускает свою очередь), чередовать писателей и читателей. Мьютексы и конд вары позволяют избежать одновременного доступа к критическим секциям из разных потоков.

Результаты

https://everest.distcomp.org/jobs/59d49033300000ab511408c6?jobId=59d4821a3000000350140862

Анализ решения

Программа удовлетворяет всем требованиям, например, в ней нет голодания читателей и писателей, есть параллельное чтение и последовательная запись, нет одновременного доступа к критическим секциям. Я думаю, её быстродействие можно ещё улучшить (Используется довольно много мьютексов и условных переменных). Так же про улучшения написано в ответе на контрольные вопросы.

Контрольные вопросы

Вопрос 1.

Измерьте время выполнения исходной и модифицированной программ на файле trace.txt. Как изменилось время выполнения? Почему? От каких факторов зависит то, насколько лучше работает модифицированная версия в сравнении с исходной? (вес 0.3)

Время до изменения: 30.043 секунд

Время после изменения: 12.590 секунд

Видно, что после изменения кода, время значительно уменьшилось. Это происходит из-за того, что мы увеличили количество кода, который может выполняться параллельно. По сути, раньше у нас вся программа могла выполняться только последовательно: либо писатель писал, либо читатель читал. При этом если читатель читал и приходило более одного читателя, все они стояли и ждали, пока тот читатель закончит. Конкретно в данной задаче это не требуется. Несколько читателей абсолютно нормально могут читать, независимо друг от друга. В решении этой проблемы и состояла вся задача.

После изменения кода, чтение стало не блокирующим для других чтений и операции чтения стали выполняться одновременно для других операций чтения. Это и ускорило выполнение.

Хочется заметить, что запись всё ещё осталась блокирующей, иначе это привело бы к возможности повредить данные. Также операции чтения и записи всё ещё не могут происходить одновременно, иначе это могло бы привести к получению не валидных данных.

Улучшение зависит от колличества кода, которое удалось распараллелить, от того как используются мьютексы и конд вары. От стратегии, которая используется в решении.

Вопрос 2.

Предположим, что в момент окончания работы со счетчиком потока-писателя доступа ожидают как писатели, так читатели. Кого следует «разбудить» в первую очередь? Почему? Можно ли это подтвердить экспериментально? (вес 0.35)

Всё зависит от поставленной задачи. Конкретно в данном случае у нас проверялись тесты на голодающих писателей и читателей. Это значит, что нам требуется, чтобы и писатели, и читатели не стояли в очереди слишком долго и не пропускали других вперед всё время.

Предположим, что при ожидающих писателях и читателях мы всегда пропускаем вперед читателей. Если читателей намного больше чем писателей или их запросы приходят чаще, писатели могут вообще никогда не проснуться, всегда уступая свою очередь.

Аналогично может быть и симметричная ситуация с большим количеством быстрых и частых писателей.

Подтверить это можно, и это было подтверждено при отсылке в *Эверест*. Моя изначальная версия программы всегда пропускала вперед читателей. Писатели не могли начать работать, следовательно на тесте starve writers происходило слишком большое время работы.

Также в процессе исправления голодающих писателей я умудрился получить в одной из посылок и голодающих читателей, отдав весь приоритет писателям. Читатели выполнялись только в том случае, если в очереди совсем не было писателей.

В качестве финального решения я использовал следующий вариант: отработавший писатель будит читателя и все читатели начинают читать. Отработавший читатель, если он последний, будит писателя. При этом возникает проблема: во время работы читателей,

писатели не могут начать работать, а новые читатели могут. Получается состояние starve writers, писатели всегда уступали место. Оно было решено следующим образом: пока есть ожидающий писатель, новые читатели становятся в очередь. Голодание читателей при этом не наступит из-за того, что все читатели, стоявшие в очереди и накопившиеся за время работы писателя, будут разбужены следующими. Голодание писателей не будет, так как последний читатель (он найдется, потому что новые больше не поступают) разбудит писателя, если он есть в очереди.

Вопрос 3.

Могут ли в предложенной вами реализации некоторые потоки бесконечно ожидать доступа к счетчику? Почему? Если да, то как можно устранить эту проблему? (вес 0.35)

В моей реализации могут. Предположим, например, всегда есть хотя бы 2 ожидающих писателя (запросов чтения и записи очень много, больше чем компьютер может обработать). Тогда может так сложиться, что одному писателю критически не везет. Он не успевает заблокировать мьютекс на запись и остается не выполненным (такая вероятность мала, но она всё же есть). Это значит, что он никогда не получит доступ к счетчику.

Решить эту проблему можно сохраняя порядок обработки запросов в соответствии с их поступлением. То есть, чем раньше запрос пришел, тем раньше он получит доступ к счетчику. Это сводит на нет всю возможность чьего-нибудь “невезения”.

Хотя это сделать можно, в данном задании этого не требовалось.