

6. Модель памяти

Илья Игашов

25 мая 2017 г.

Задача. *TASSpinlock*

Решение.

- 1) Работаем с одной атомарной переменной `locked_`.
- 2) В модели `std::memory_order_seq_cst` для любых двух потоков события чтения из переменной `locked_` в одном потоке и записи в нее в другом потоке упорядочены частичным порядком `synchronizes-with`. Поскольку никаких других обращений к памяти в этом коде нет, по тривиальному транзитивному замыканию легко определяется порядок `happens-before`.
- 3) Можно ослабить гарантии упорядочивания до `release/acquire`, поскольку все потоки синхронизируются через один атомик, следовательно все приведенные выше рассуждения справедливы для этой модели памяти. `Relaxed` недостаточно, поскольку в этом случае не возникает стрелка `synchronizes-with`, а нам это нужно для построения порядка `happens-before`.

`Happens-before` необходим для того, чтобы показать, что разные потоки видят необходимые записи в переменные, которые были произведены другими потоками. В частности, поток, вошедший в критическую секцию, увидит все записи в разделяемую память потока, который был в критической секции до него: это как раз достигается за счет построенного порядка `happens-before`.

Задача. *SPSCRingBuffer*

Решение.

- 1) Вектор `buffer_` является неатомарным. В методе `Publish` происходит запись в вектор, а в методе `Consume` – чтение из него. Порядок чтения и записи в конкретную ячейку вектора важен. Этот порядок гарантируется отношением `happens-before`, который возникает из отношений `program-order` и `synchronized-with`.

Локальные переменные `curr_head` и `curr_tail` также являются неатомарными. Внутри секции

```
const size_t curr_head = head_.load(/* memory order */);  
const size_t curr_tail = tail_.load(/* memory_order */);
```

порядок не важен, поскольку эти две записи друг от друга не зависят. Далее важно, чтобы последующие чтения этих переменных происходили после первоначальной записи в них. Это гарантируется отношением `program-order`.

- 2) Рассмотрим атомарную переменную `head_`. операции чтения и записи в эту переменную связаны отношением `synchronized-with`. То же самое и с переменной `tail_`. Учитывая порядок `program-order`, определенный внутри каждого потока (в коде программы), видим, что алгоритм гарантирует отношение `happens-before`. Например, фрагмент из `Consume` в потоке В

```
const size_t curr_tail = tail_.load(/* memory order */);  
  
if (Empty(curr_head, curr_tail)) {  
    return false;  
}
```

видит запись в переменную `tail_` в предшествующем `Publish` в потоке `A` как раз благодаря `happens-before`.

- 3) Хотя у нас имеются два атома, важно, чтобы синхронизация происходила на каждом атоме отдельно, а именно, читающий поток должен видеть последнюю запись (синхронизация через атомик `tail_`), и пишущий поток должен видеть последнее чтение (синхронизация через атомик `head_`). Глобальный порядок на обоих атомах не требуется, однако на каждом по отдельности он нужен, следовательно, можно ослабить гарантии упорядочивания до `acquire/release`.

Задача. *LazyValue*

Решение.

- 1) Внутри метода `Get` используется неатомарная переменная `curr_ptr`. Важно, чтобы вначале мы записали в нее значение атомарной переменной `ptr_to_value_`, и только потом уже работали с ней. Внутри потока это выполняется благодаря `program-order`, все нормально.
- 2) Идея в том, чтобы не создавать уже созданный объект. Сначала мы запишем в `curr_ptr` значение, которое атомарно прочитаем из `ptr_to_value_`. И дальше уже в зависимости от этого значения будем или не будем создавать объект (само создание происходит после захвата мьютекса и после проверки условия, что объект еще не создан). Поскольку все операции с `ptr_to_value_` атомарны и на них определено отношение `synchronized-with`, а внутри одного потока еще и работает `program-order`, то мы видим, что алгоритм гарантирует `happens-before`. Например, после захвата мьютекса создание объекта произойдет в случае, если мы прочитали из `curr_ptr` `nullptr`. Это произойдет, если мы прочитали из `ptr_to_value_` `nullptr` (`program-order`). А это чтение, в свою очередь, видит последнюю запись в `ptr_to_value_` (`synchronized-with`), значит по транзитивности имеем стрелку `happens-before`.
- 3) Можно ослабить гарантию упорядочивания в первом чтении `ptr_to_value_` в методе `Get` до `acquire/release`, так как мы синхронизируемся на одном атоме, а в этой модели все еще работает `synchronized-with`. Этот порядок нам нужен, чтобы два потока не создали объект два раза, а значит `relaxed` не подходит. Остальные чтения и записи в `ptr_to_value_` в методе `Get` происходят внутри критической секции – тут можно ослабить гарантии до `relaxed`, потому что они ни на что не влияют: все это происходит под мьютексом.