Задание №7. Линеаризуемость

Рубаненко Евгений

Май 2017

1 Распределенный счетчик

1.1 Пункт 1

Приведем пример конкурентной истории, которая никак не может быть представлена в виде последовательных исполнений функций (Откуда сразу будет следовать, что такой счетчик не является линеаризуемым). Представим, что у нас есть три потока. Первый делает $Get()$, второй делает $Add(2)$, а третий - $Add(3)$. Изначально во всех ячейках массива записаны нули. Допустим, что первый поток начал исполняться и уже успел прочитать и 0 из первой ячейки массива, и 0 из второй ячейки массива, а после уснул. Далее предположим, что был вызван $Add(2)$ (и вызов завершился) и $Add(3)$ (также завершился). Потом первый поток проснется и вызов $Get()$ вернет 3.
Вызовы $Add(2)$ и $Add(3)$ в конкурентной истории упорядочены. Отсюда следует, что при последовательном исполнении они останутся в таком же порядке. Тогда нам осталось понять, в какой момент относительно этих двух операций завершается вызов $Get()$. Если он завершится раньше, чем все вызовы $Add(something)$, то он вернет 0. Если между ними - то должен вернуть 2. Если после - 5. Получаем противоречие.

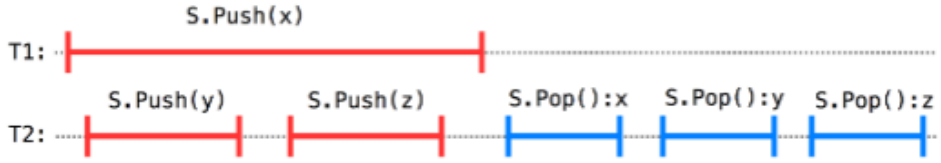
2 Поиск без блокировок в оптимистичном сортированном списке

Покажем, что чтение флага $marked$ нельзя брать в качестве точки линеаризации в $Contains(x)$. Допустим, что изначально в списке элемент x присутствовал. Далее был произведен вызов $Contains(x)$. Допустим, что данный вызов проверил первое условие в $return$, но не успел проверить второе (чтение $marked$) и уснул. Затем пришел второй поток, позвал $Remove(x)$, этот вызов успешно завершился. Теперь $marked$ равен $true$. Затем этот же поток позвал $Insert(x)$, и этот вызов тоже завершился. Теперь в списке есть новый узел, в котором лежит элемент x. Затем первый поток просыпается, читает $marked$ и возвращает $false$.
В конкурентной истории сначала завершается вызов $Remove(x)$, а потом $Insert(x)$. Тогда в последовательном исполнении они тоже будут выполнены в соответсвующем порядке. Так как мы выбрали в качестве точки линеаризации чтение $marked$, а оно происходит после заверше-

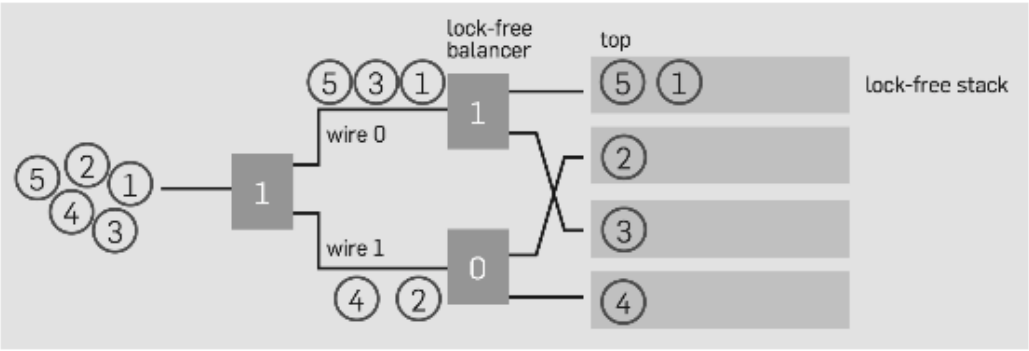
ния вызовов $Remove(x)$ и $Insert(x)$, то в последовательном исполнении $Contains(x)$ должен быть выполнен после $Remove(x)$ и $Insert(x)$. Но из такого исполнения получается, что $Contains(x)$ должен вернуть $true$ - противоречие.

3 Стек с балансирующим деревом

Покажем, как именно возникает нелинеаризуемая история в "стеке" с балансирующим деревом.
История:
Допустим, что у нас есть 2 потока. Сначала первый поток вставляет элемент x. Затем второй поток вставляет элементы y и z. Далее второй поток вынимает из "стека" элементы (в написанном порядке): x, y, z. (С точки зрения продолжтельности это выглядит так, как на картинке)



Очевидно, что такая история невозможна с точки зрения последовательного исполнения методов $Push()$ и $Pop()$: во втором потоке вызовы упорядочены, а значит, достать из "стека" y раньше z невозможно - мы ведь положили z позже.
Но в действительности история имеет место.



0) Изначально в "стеке" ничего нет, значения всех узлов равны 0
1) Первый поток делает $Push(x)$; вызов зависает между двумя узлами дерева, первый бит теперь равен 1

2) Второй поток делает $Push(y)$; вызов завершается, знаечния узлов есть соответственно 0 и 1 (значение нижнего узла)
3) Второй поток делает $Push(z)$; вызов завершается (он обгоняет $Push(x)$); значения узлов теперь 1 и 1 (верхний)
4) Вызов $Push(x)$ первого потока завершается
Итак, значения узлов есть 1, 0 (верхний) и 1 (нижний). Теперь уже понятно, что три вызова $Pop()$ из второго потока вернут элементы x, y, z в указанном порядке.