Задача. *B. Tricky Mutex. Дана реализацию мьютекса на RMW-операциях инкремента и декремента. Гарантирует ли эта реализация взаимное исключение (mutual exclusion) и свободу от взаимной блокировки (deadlock freedom)?*

Решение. При числе потоков равном одному, обе гарантии, очевидно, выполняются. Далее потоков больше одного.

Покажем, что данная реализация гарантирует взаимное исключение.

Доказательство.

Предположим противное, пусть два потока A и B одновременно находятся в критической секции. Оба они должны были пройти цикл ожидания, то есть сделать $fetch\_add$ и операцию сравнения результата с 0. Для определенности пусть поток A сделал $fetch\_add$ раньше потока B(эта операция атомарна, поэтому никаких проблем не возникает). Раз оба потока прошли в критическую секцию, значит результат $fetch\_add$ у обоих был ≤ 0(условие невхождения в тело цикла while). Далее рассмотрим 2 случая.

1) Если в критической секции уже есть потоки, то они все инкрементировали счетчик при входе, а декрементировать еще не успели(т.к. они еще внутри критической секции), значит результаты операций $fetch\_add$ и у A, и у B должны были быть > 0, противоречие.

2) Если в критической секции не было других потоков, то т.к. A сделал $fetch\_add$ первым(то есть сделал значение счетчика равным 1), а в дальнейшем, пока A не выйдет из критической секции, значение счетчика никак не станет меньше 1, то когда B сделает операцию $fetch\_add$ ее результат будет ≥ 1, а значит, он не сможет зайти в критическую секцию, противоречие.

Таким образом мы получили, что в каждый момент в критической секции может находиться не более одного потока, а значит гарантируется взаимное исключение. Что и требовалось доказать.

Теперь рассмотрим свободу от взаимной блокировки.

Рассмотрим пример.

Пусть у нас есть два потока A и B и счетчик $thread\_count$. В начальный момент времени, пусть поток A находится в критической секции($thread\_count = 1$). Будем управлять планировщиком. Поток B хочет зайти в критическую секцию$\rightarrow$ $B : fetch\_add(1)(thread\_count = 2.)\rightarrow$ переключаем планировщик, и потом ни разу не включая его на B, включим на A, когда он выходит из критической секции$\rightarrow$ $A : fetch\_sub(1)(thread\_count = 1)\rightarrow$ и пусть теперь какой-то поток, для определенности тот-же A, опять захотел войти в критическую секцию$\rightarrow$ переключим планировщик на него$\rightarrow$ $A : fetch\_add(1)(thread\_count = 2)\rightarrow$ переключим планировщик опять на $B \rightarrow B : fetch\_sub(1)(thread\_count = 1)$, но B не получил доступа в критическую секцию, поэтому $B : fetch\_add(1) \rightarrow$ опять переключим планировщик на $A \rightarrow A : fetch\_sub(1)$ $(thread\_count = 1)$ но он тоже не получил доступа к критической секции, поэтому $A : fetch\_add(1)$ $(thread\_count = 2) \rightarrow$ переключим планировщик на $B \rightarrow B : fetch\_sub(1)$ $(thread\_count = 1)$, но он снова не получил доступа, значит $B : fetch\_add(1)$ $(thread\_count = 2) \rightarrow$ опять переключаем планировщик на A и продолжаем переключать его в таком порядке. (На следующей странице приведена таблица1 иллюстрирующая вышесказанное).

При таком порядке переключения планировщика ни один из потоков A и B, которые оба хотят попасть в критическую секцию, в нее никогда не попадут, а значит свобода от взаимной блокировки не гарантируется в этой реализации.

Таблица 1: Действия потоков A и B и счетчик, после их выполнения.

thread _count	1	2	1	2	1	2	1	2	1
A	+1		-1	+1			-1	+1	
B		+1			-1	+1			-1

Данная реализация гарантирует взаимное исключение (mutual exclusion), но не грантирует свободу от взаимной блокировки (deadlock freedom).