B. Tricky Mutex - 3 балла

Рассмотрим реализацию мьютекса на RMW-операциях инкремента и декремента:

```
class tricky_mutex {
public:
  void lock() {
      while (thread_count.fetch_add(1) > 0) {
          thread_count.fetch_sub(1);
      }
  }

  void unlock() {
      thread_count.fetch_sub(1);
  }

private:
  std::atomic_int thread_count = 0;
};
```

Гарантирует ли эта реализация взаимное исключение (mutual exclusion) и свободу от взаимной блокировки (deadlock freedom)?

Ответ: Заметим, что счетчик thread_count, в процессе выполнения программы, не может принимать отрицательные значения.

Докажем это. Допустим, что thread_count в некоторый момент времени отрицателен. Это означает, что инструкций thread_count.fetch_sub(1) было больше, чем инструкций thread_count.fetch_add(1), чего не может быть, так как за каждой инструкцией инкрементирования следует инструкция декрементирования, вне зависимости от того, вошел поток в критическую секцию или нет. Значит, в любой момент выполнения программы количество выполненных инструкций инкремента больше либо равно количеству выполненых инструкций декремента. Получаем противоречие и thread_count >= 0 в любой момент времени.

Докажем, что TrickyMutex гарантирует взаимное исключение. Допустим это не так, и в некоторый момент выполнения программы два потока, назовем их A и B, попали в критическую секцию. Это значит, что перед проверкой в потоке B условия в while переменная thread_count приниала значения <= 0. Но, как было доказано выше, thread_count >= 0. Значит, перед проверкой в потоке B условия в while переменная thread_count == 0. Но в этот момент поток A находился в критической секции. Это означает, что потоком A было выполнено инкрементирование переменной thread_count, но не выполнено его декрементирование. Это означает, что количество выполненых инструкций thread_count.fetch_add(1) строго больше количества инструкций thread_count.fetch_sub(), что означает, что thread_count > 0. Противоречие. Значит в критической секции может быть лишь один поток, то есть гарантируется взаимное исключение.

Докажем, что TrickyMutex не гарантирует свободу от взаимной блокировки.

Пример:

Допустим, что поток A начинает освобождать мьютекс, но еще не выполнил инструкцию декрементирования. Далее приводится пример последовательного

исполнения инструкций 3 потоков.

A	B	C	
1) unlock()			*thread_count == 1*
2) thread_count.fetch_add(1)			*thread_count == 2*
3) thread_count.fetch_sub(1)			*thread_count == 1*
4) thread_count.fetch_add(1)			*thread_count == 2*
5) thread_count.fetch_add(1)			*thread_count == 3*
6) thread_count - 1 > 0			
7) thread_count.fetch_sub(1)			*thread_count == 2*
8) thread_count.fetch_add(1)			*thread_count == 3*
9) thread_count - 1 > 0			
10) thread_count.fetch_sub(1)			*thread_count == 2*

11) thread_count.fetch_add(1) *thread_count == 3*
12) thread_count - 1 > 0
13) thread_count.fetch_sub(1) *thread_count == 2*
14) thread_count.fetch_add(1) *thread_count == 3*

Далее последовательность действий 6-14 повторяются, таким образом образуя deadlock.