Взаимное исключение

A. Алгоритм Петерсона - 2 балла

Поток 0	Поток 1

want[0] = true
victim = 0
while (want[1] and victim == 0)
    // wait
	want[1] = true
	victim = 1
	while (want[0] and victim == 1)
	    // wait

Будет ли алгоритм Петерсона для двух потоков гарантировать взаимное исключение, если в doorway-секции метода lock поменять местами взвод флажка (want[thread_index].store(true)) и запись в victim (victim.store(thread_index))?

Если будет, то приведите доказательство корректности. Если нет, то приведите пример последовательного исполнения в модели чередования, в котором захват мьютекса из двух потоков приводит к нарушению взаимного исключения.

B. Tricky Mutex - 2 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? Гарантирует ли она свободу от взаимной блокировки (отсутствие дедлоков и лайвлоков)?

Будем предполагать, что планировщик не может навечно заморозить исполнение какого-либо потока, если у того еще есть инструкции, которые он хочет выполнить.

C. Метод try_lock для ticket спинлока - 2 балла

Семантика метода try_lock: захватить мьютекс без ожидания (если это возможно) и вернуть true; если попытка захвата не удалась, и потоку нужно ждать, то вернуть false.

Неудачные вызовы try_lock не должны блокировать вызовы lock/unlock в других потоках.

Придумайте реализацию метода try_lock для ticket спинлока. Методы lock и unlock должны остаться без изменений.

В качестве разогревающего упражнения - придумайте реализацию try_lock для test-and-set спинлока.

D. Ticket Spinlock - 2 балла

В отличие от Test-And-Set спинлока, который допускает голодание потоков, Ticket спинлок гарантирует честность: потоки завладеют блокировкой в том порядке, в котором они получили номерки от атомарного счетчика.

Но у честности есть и обратная сторона:

Если наращивать число потоков, которые соревнуются за ticket lock, то в какой-то момент число критических секций в единицу времени сильно упадет. Почему так происходит и сколько потоков для этого нужно?

Подсказка: подумайте, как ведет себя TAS и Ticket спинлоки в случае, когда ожидающий спинлока поток вытесняется планировщиком.

```
void ticket_spinlock::lock() {
   const ticket_t this_thread_ticket =
next_free_ticket_.fetch_add(1);
   while (this_thread_ticket != owner_ticket_.load()) {
       // wait
   }
}
```

```
void tas_spinlock::lock() {
   while (locked_.exchange(true)) {
       // wait
   }
}
```

Подсказка: подумайте, как ведет себя TAS и Ticket спинлоки в случае, когда ожидающий спинлока поток вытесняется планировщиком.

E. Tournament Tree Mutex - 4 балла

Гарантирует ли tournament tree mutex честность (fairness)? Верно ли, что поток, который раньше подошел к мьютексу, раньше войдет в критическую секцию?

Гарантирует ли tournament tree mutex свободу от голодания?

Реализуйте древесный мьютекс из мьютексов Петерсона для n потоков.

Конструктор должен принимать число потоков (num_threads), методы lock и unlock - номер потока (thread_index)

```
tree_mutex(std::size_t num_threads)
```

Методы:
```
void lock(size_t thread_index)
void unlock(size_t thread_index)
```

Параметр thread_index принимает значения от 0 до num_threads-1.

Храните бинарное дерево в виде линейной развертки, как в реализации двоичной кучи/пирамиды.

Помните, что алгоритм Петерсона корректно работает для потоков с номерами 0 и 1, в то время как потоки древесного мьютекса имеют номера от 0 до n-1.

Можно считать, что мьютекс будет использоваться корректно, т.е. mtx.unlock(t) будет вызван только если поток с индексом t владел мьютексом.

В секции ожидания в мьютексе Петерсона вызывайте std::this_thread::yield(), который будет переключать текущее ядро на выполнение другого потока. В противном случае при исполнении на одном ядре поток, ждущий в wait-секции, будет целый квант времени греть воздух и не давать другим потокам исполняться, в том числе и тому потоку, который в данный момент владеет мьютексом.

F. Взаимное исключение в распределенных системах - 3 балла

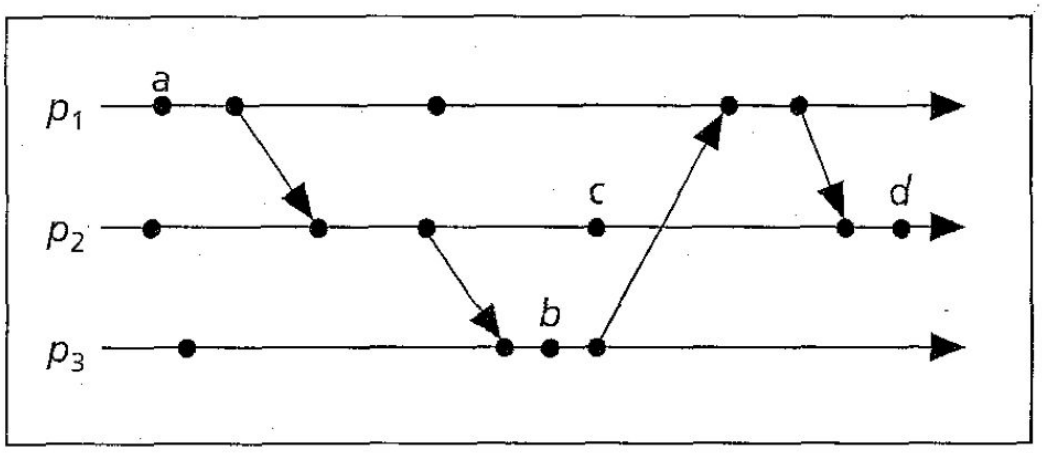
В многопроцессорной системе с разделяемой памятью мы моделируем исполнения в **модели чередования**: единственный процессор по очереди выполняет инструкции в порядке их следования в программе, переключаясь с потока на поток.

Взаимное исключение в такой модели определить довольно просто: если после очередного шага процессора два потока считают, что они вошли в критическую секцию, то взаимное исключение нарушено.

Рассмотрим теперь модель **обмена сообщениями** (**message passing**), которая используется в распределенных системах:

- взаимодействующие агенты называются **процессами**
- процессы коммуницируют только с помощью **обмена сообщений**
- **нет глобальных часов, нет глобального порядка событий**

Исполнения можно моделировать следующим образом:



Каждая новая точка на таймлайне процесса - это:

- Переход из одного внутреннего состояния в другое (этот переход не наблюдают другие процессы)
- Отправка сообщения другому процессу
- Получение сообщения, посланного другим процессом

Сделаем несколько предположений:

- Сеть **надежна**, никогда не теряет отправленные сообщения.
- Сеть работает **асинхронно**, никаких гарантий на время доставки нет. Известно лишь, что каждое отправленное сообщение будет получено адресатом, но неизвестно, когда именно это случится.

Можно считать, что время на разных таймлайнах течет с разной скоростью. Глобального порядка на событиях нет, синхронизированных часов нет.

Например, на рисунке выше невозможно установить, какое событие произошло раньше - b или c.

В такой модели мы хотим придумать протокол взаимного исключения.

Цель, которую преследует каждый процесс - перейти в выделенное состояние "нахожусь в критической секции". Причем одновременно в таком состоянии может находиться не более одного процесса.

Протокол взаимного исключения должен строиться на сообщениях:

Если в многопроцессорной системе агент (поток) при захвате мьютекса опирался на прочитанные значения из разделяемых ячеек памяти (вспомним протоколы Петерсона или Лампорта), то теперь он (процесс) принимает решение только на основе сообщений, полученных от других процессов, никаких других средств коммуникации нет.

В многопроцессорной системе мы говорили, что потоки не должны одновременно находиться в критической секции. Но что такое **одновременно** в распределенной системе? Глобального порядка событий нет, никаких гарантий на синхронизацию часов нет.

Как нам (внешнему наблюдателю) понять, что взаимное исключение нарушено? Как формализовать требование взаимного исключения в модели обмена сообщениями?

Критерий взаимного исключения конечно же не должно зависеть от конкретного протокола/алгоритма.

Для решения задачи не нужно знать или самому придумывать протокол взаимного исключения в модели обмена сообщениями, это непростая задача (хотя и не слишком сложная). Нужно лишь придумать критерий взаимного исключения.

Подсказка:

Критические секции должны быть линейно упорядочены друг относительно друга во времени, даже несмотря на отсутствие глобального порядка на всех событиях в системе. Поэтому сначала определите понятие “последовательно”, а потом уже “параллельно” или “одновременно”.