## 1. В чем недостаток организации взаимоисключения работы критических секций на основе использования блокирующей глобальной переменной (замка)? для приведенного алгоритма отпишите ситуацию, когда его использование будет приводить к нежелательному развитию событий(в чем проблема).

```
shared int lock = 0;
  while(lock); lock = 1;
critical section
lock = 0;
```

Такое решение не удовлетворяет условию взаимоисключения, так как действие while(lock); lock = 1 ; не является атомарным. Допустим, что поток P0 протестировал значение переменной lock и принял решение двигаться дальше. В этот момент, еще до присваивания переменной lock значения 1, планировщик передал управление потоку P1. Он тоже изучает содержимое переменной lock и тоже принимает решение войти в критический участок. Мы получаем два процесса, одновременно выполняющих свои критические секции.

## 2. В каком случае целесообразно использовать алгоритм строгого чередования, приведенный ниже, а в каком случае его использовать невозможно?

```
shared int turn = 0;
  while (turn != i);
    critical section;
  turn = 1-i;
```

Алгоритм не удовлетворяет условию прогресса. Например, если значение turn равно 1 и процесс P0 готов войти в критический участок, он не может сделать этого, даже если процесс P1 находится в remainder section.

Недостаток предыдущего алгоритма заключается в том, что процессы ничего не знают о состоянии друг друга в текущий момент времени.

## 3. какого рода проблемы и в какой момент исполнения могут возникнуть при использовании приведенного ниже алгоритма флагов готовности.

```
shared int ready[2] = {0, 0};
  ready[i] = 1;
  while(ready[1-i]);
    critical section
  ready[i] = 0;
```

алгоритм обеспечивает взаимоисключение, позволяет процессу, готовому к входу в критический участок, войти в него сразу после завершения эпилога в другом процессе, но все равно нарушает условие прогресса. Пусть процессы практически одновременно подошли к выполнению пролога. После выполнения присваивания ready[0] = 1 планировщик передал процессор от процесса 0 процессу 1, который также выполнил присваивание ready[1] = 1. После этого оба процесса бесконечно долго ждут друг друга на входе в критическую секцию. Возникает ситуация, которую принято называть тупиковой (deadlock).

## 4. Сформулируйте нормальные определения операций P(S) и V(s) над переменной-семафором S.

**Двоичный семафор (binary semaphore)** – переменная S, которая может находиться в двух состояниях: "открыт" и "закрыт"; над S определены две операции ( "семафорные скобки"): P(S) – закрыть, V(S) – открыть. При попытке закрыть уже закрытый семафор происходит прерывание, и ОС добавляет текущий процесс в очередь к закрытому семафору. Операция V(S) активизирует первый стоящий в очереди к S процесс, который успешно завершает операцию P(S). Если семафор S уже открыт, операция V(S) не имеет никакого эффекта.

## 5. Перечислите условия возникновения тупиков.

1. Условие взаимоисключения (Mutual exclusion). Одновременно использовать ресурс может только один процесс.
2. Условие ожидания ресурсов (Hold and wait). Процессы удерживают ресурсы, уже выделенные им, и могут запрашивать другие ресурсы.
3. Условие неперераспределяемости (No preemtion). Ресурс, выделенный ранее, не может быть принудительно забран у процесса.
   Освобождены они могут быть только процессом, который их удерживает.
4. Условие кругового ожидания (Circular wait). Существует кольцевая цепь процессов, в которой каждый процесс ждет доступа к ресурсу, удерживаемому другим процессом цепи.

Для образования тупика  необходимым и достаточным является выполнение **всех четырех** условий.