Quiz 1

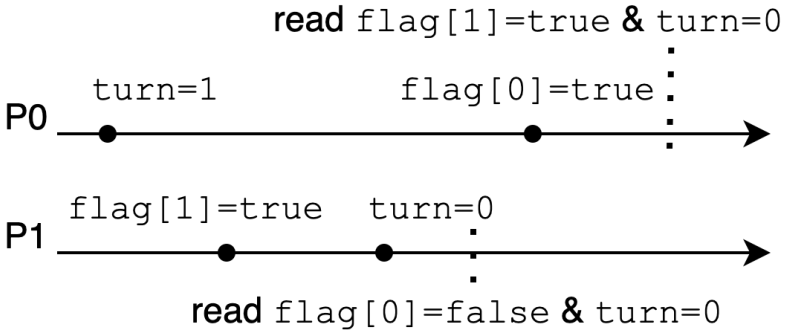
Sept. 2023

1. What if we reverse the order of the first two lines in the 2-process Peterson's algorithm (for one of the processes)? Would it work? What if we get rid of flag variables (only use turn as a shared variable)?

   If we change the first two lines in P0 such as:

   ```
   P0:
   turn = 1;
   flag[0] = true;
   // others do not change
   ```

   then, for the time line of two threads could be like:



   and finally, both P0 and P1 will enter into the critical section.

   If we ignore `flags` and only use `turn`, this would not happen, because we only modify `turn` once in an execution of P0 or P1, and whenever we read `turn` in either P0 or P1, the other will enter into `while` part and wait.

2. Prove that Tournament algorithm ensures: (a) mutual exclusion: no two processes are in the critical section at a time; (b) starvation-freedom: every process in the trying section eventually reaches the critical section (assuming no process fails in the trying, critical, or exit sections)

   (a) Tournament algorithm is built like a binary tree. In the competition of two processes, only one can enter to the higher level, which means there's impossible for both two processes enter into next level. Similarly, for the root node, which is the final process for the critical section, could only exist one process.
   (b) The level of the tree is finite, which means even if a process "loses" in the competition, it still have chance to "compete" (because no one fails), and compare with last time, the process cannot "back", so it rests in the present level or moves into next level. And finally all the process could reach to the critical section.

3. Let S be a safety property. Show that if all finite runs of an implementation I are safe (belong to S) then all runs of I are safe

   Assume all the finite runs have the property P belong to S, what we want is to show all the infinite runs also have the property P.

For all prefix σ_i in all finite runs, $\sigma_i \in P$. $\forall \sigma_i \in P$ and P is limit-closed, so $\lim_{i\to\infty} \sigma_i \in P$.
Therefore, all infinite runs satisfy the property P.

4. Show that every property is an intersection of a safety property and a liveness property

For a property P, what we want is to find S_p and L_p satisfy $P = S_p \cap L_p$.
Assume $S_p = S$, which means set of the results won't happen for P. According to the definition of L, which will include the good result that would happen, thus L oppose to be as large as possible, assume $L_p = P \cup S^C$. $S_p \cap L_p = S \cap (P \cup S^C) = P$.

5. Show that every unsafe run σ has an unsafe finite prefix σ': every extension of σ' is unsafe

By contradictory, assume σ has safe prefix, i.e. $\sigma_i \in P$, and use the property of limit-closed, $\lim_{i\to\infty} \sigma_i \in P$, so that σ' is safe, which shows a contradiction. Therefore, every unsafe run σ has an unsafe finite prefix σ': every extension of σ' is unsafe.

6. Show that linearizability is compositional: A history H on AxB is linearizable if and only if H|A and H|B (histories restricted to the events on objects A and B, respectively) are linearizable

What we want is to show that H is linearizable ⇔ both H|A and H|B are linearizable.
From left to right: Obviously, $A \subset H$ and $B \subset H$, therefore H is linearisable ⇒ both H|A and H|B are linearisable.
From right to left: For A and B, R_A and R_B are the set of the sequential responses respective to A and B. Let H' be the history of $R_A \cup R_B$. Complete(H') could be equivalent to a legal history of sequence S and also S reserves the sequential relation in H, thus H could be linearizable.

7. Show how the elements of the "periodic table of progress" are related to each other. (Property P is weaker than property P' if P' is a subset of P.)

stronger ——— weaker
WF <— LF <— OF
LF <— DF
<— SF <— DF

8. Show that the sequential queue implementation presented on slide 48 is linearizable and wait-free, *as is, without locks*, if used by two processes: one performing only enqueue operations and one performing only dequeue operation. (You need also to find the matching enqueue implementation.)

It could be linearizable with the sequence `q.enq() q.deq()`.
An algorithm of `enq()` with wait-free and without locks as below:

```
shared
      Node tail, head;
Node
      value;
      next;
enq()
      Node last := tail;
      Node next := last.next;
      if ( last = tail )
            Add and Update new Node
      else
            tail.last = next;
```

9. Devise a simple queue implementation shared by any number of processes in which enqueue and dequeue operations can run concurrently (data races between these operations are allowed).

```
Node
      data;
      next;
shared
      Node head;
      Node tail;
enq()
      new Node node;
      node.data := data;
      node.next := null;
      tail := node;
      tail.next := null;
deq()
      if tail != head :
            Node node := head;
            head := head.next;
            return node;
      return null;
```

`enq()` and `deq()` only do the operations to tail and head respectively, which could allow the data races between these operations.