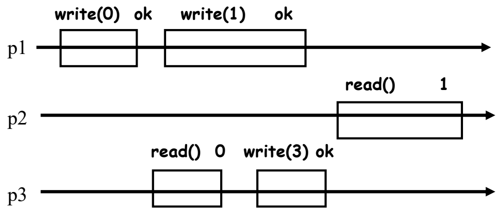
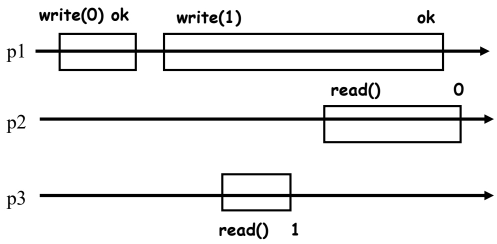
Problem 1 (9 points)

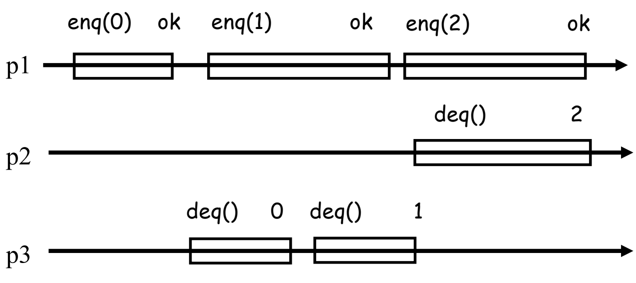
- Can the history below be exported by an *atomic* register? (Yes/No) If yes, assign a linearization point to each operation.



- Can the history below be exported by an *atomic* register? (Yes/No) If yes, assign a linearization point to each operation.



- Is the history below linearizable with respect to the *FIFO queue* specification? (Yes/No) If yes, assign a linearization point to each operation.



Problem 2 (5 points)

Consider the 2-process Peterson's mutual exclusion algorithm:

```
bool flag[0]   = false;
bool flag[1]   = false;
int turn;
```

```
P0:                                P1:

flag[0] = true;                    flag[1] = true;
turn = 1;                          turn = 0;
while (flag[1] and turn==1)        while (flag[0] and turn==0)
{                                  {
          // busy wait                       // busy wait
}                                  }
// critical section                // critical section
…                                  …
// end of critical section         // end of critical section
flag[0] = false;                   flag[1] = false;
```

Suppose that p_0 executes the first two lines of its algorithm in the reverse order:

1. turn = 1;
2. flag[0] = true;

Prove that the resulting algorithm is not correct.

Problem 3 (6 points)

We say that a property P *is stronger than* a property P' if $P \subseteq P'$, i.e., every run that satisfies P also satisfies P'.

Recall the two properties:

- *SF* (*starvation-freedom*): if every process is correct, then every process makes progress.
- *LF* (*lock-freedom*): at least one correct process makes progress.

What is the relation between *SF* and *LF*?

Problem 1 (4 points)

Classify the following properties into safety/liveness. If a property is an intersection of the two, specify the corresponding safety and liveness properties. Justify your answers.

- Every process eventually outputs a value.

- No two processes output different values.

- Every process eventually outputs a previously proposed input of some process or crashes (stops taking steps).

- No two correct processes output different values.

Problem 2 (4 points)

We say that a property P *is stronger than* a property P if $P \subseteq P'$. What is the relation between *starvation-freedom* (SF) and *lock-freedom* (LF)? Explain why.

Problem 3 (4 points)

Give an algorithm that implements a safe 1WNR M-valued register (for some fixed M) using $\lceil \log M \rceil$ safe 1WNR binary registers. Provide a proof of correctness.

If we replace the safe binary registers with *regular* ones, do we get a *regular* M-valued register implementation?

Problem 3: Linked Lists (3 points)

In the *validate* function of the lazy linked-list implementation (cf. the next page), is checking `curr.marked` really necessary? Justify your answer.

Lazy synchronization: logical removals and wait-free contains

```
private boolean validate(Node pred, Node
    curr) {

  return !pred.marked && !curr.marked &&
          pred.next==curr;
}
```

- remove first marks the node for deletion and then physically removes it
- contains returns true iff the node is reachable and not marked
- A node is in the set iff it is an unmarked reachable node

```
public boolean remove(int item)
   while (true){
      Node pred=head;
      Node curr=pred.next;
      while (curr.key<item){
         pred=curr;
         curr=curr.next;
      }
      pred.lock();
      try {
         curr.lock();
         try {
          if (validate(pred,curr)){
          if (curr.key!=item){
             return false;}
          curr.marked=true;
          pred.next=curr.next;
          return true;}
         } finally{
             curr.unlock(); }
      } finally{
        pred.unlock();}
   }
}
```


Lazy synchronization: wait-free contains

```
public boolean insert(int item){
   while (true){
      Node pred=head;
      Node curr=pred.next;
      while (curr.key<item){
         pred=curr;
         curr=curr.next;
      }
      pred.lock();
      try {
         curr.lock();
         try {
          if (validate(pred,curr)){
           if (curr.key==item) {
             return false;
           }
          Node node = new Node(item);
          node.next=curr;
          pred.next=node;
          return true;
         } finally{
             curr.unlock(); }
      } finally{
        pred.unlock();}
   } }
}
```

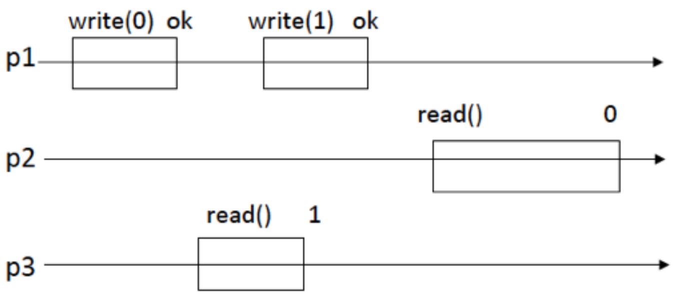
```
public boolean contains(int item){

   Node curr=head;
   while (curr.key<item){
      curr=curr.next;
   }
   return (curr.key==item)&& !curr.marked ;
}
```

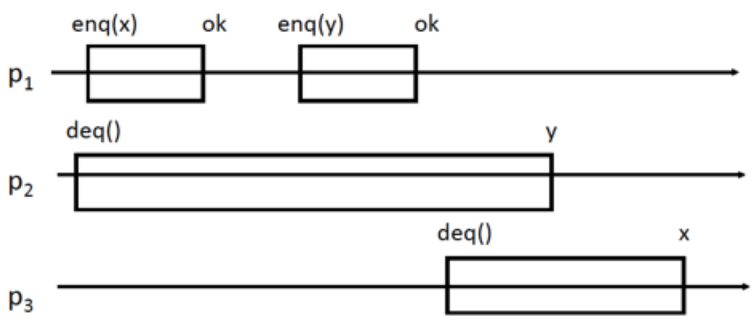
Problem 4 (4,5 points)

- Depict a history of a one-writer one-reader register that satisfies the specification of a regular register, but *does not* satisfy the specification of an atomic register.

- Is this a history of a regular register (Yes/No)? Why?



- Is the history below linearizable with respect to the specification of queue? (Yes/No) If yes, assign a linearization point to each operation.



Problem 5 (5 points)

A counter object exports one operation *inc-read*, which (in one atomic step) increments the counter and returns the old value.

Show that counter has consensus number 2:

- 2-process consensus can be solved using counters and atomic registers;
- 3-process consensus cannot be solved using counters and atomic registers.