1. Multi-threading. For each of the following programs in the language we considered in class, give the possible final values that could be stored in x.

   a. spawn p1  
      spawn p2  
      halt  
      p1: x = 1  
          halt  
      p2: x = 0  
          halt

   b. b = 1  
      spawn p1  
      spawn p2  
      halt  
      p1: if b != 0 goto p12  
          halt  
      p12: x = 1  
           halt  
      p2: if b != 0 goto p22  
          x = 2  
          halt  
      p22: halt

   c. x = 0  
      spawn p1  
      spawn p2  
      halt  
      p1: y = x  
          x = 2*y  
          halt  
      p2: z = x  
          x = z + 1  
          halt

2. Locks

   a. What values may x hold when the following program terminates?  
      x = 0  
      spawn p1  
      spawn p2  
      halt  
      p1: acquire 1  
          y = x
x = 2*y
release l
halt

p2: acquire l
    z = x
    x = z + 1
    release l
    halt

b. Explain how locks may be used to implement a new statement for our language, 
   await x with l. The meaning of this statement is as follows. The program 
   blocks until x is non-zero and l is available, then begins execution of the 
   statements immediately following with lock l locked and x initially guaranteed to 
   be non-zero. Your explanation should contain a code snippet. You may use any 
   new variables / labels you need.

c. Explain why the following program may deadlock.

    spawn p1
    spawn p2
    halt

    p1: acquire l1
        acquire l2
        release l2
        release l1
        halt

    p2: acquire l2
        acquire l1
        release l1
        release l2
        halt

d. A commonly used policy to avoid deadlocks when using locks is the following. 
   Processes must not release any locks until they have acquired all locks they need, 
   and they must attempt to lock individual locks in the same order. Explain how 
   this would fix the situation in the immediately previous problem.

3. Signaling

   a. Using wait and notifyAll, implement a three-thread system consisting of one 
      withdrawer and two depositors. Each depositor deposits $10 into an account x 
      (originally x is 0); the withdrawer may withdraw $20 as soon as the account 
      contains $20. You must ensure that account x is never overdrawn (i.e. that 
      variable x is never negative).

   b. In class we implemented wait using a waiter set; if a thread i executes wait l 
      then an insertion is made into the wait set. Explain how busy-waiting (use of
while loops to emulate waiting) may instead be used to implement `wait`. How would `notifyAll` be implemented in this case?

c. The term *busy waiting* refers to a style of programming in which threads remain in loops, testing for the availability of a condition or shared resource. Explain why the use of locks / `wait` / `notifyAll` is preferable to busy-waiting.