Problem 1.

Maintain a queue on each process. Add the vector clocks on each process to the queue from the left to the right.

When token reaches a certain process Node, check if the process is interested in accessing the shared resource. If it’s not interest, just pass the token to the next node. If it is interested. Send a request to other process node to compare its vector on the front of the queue with other processes’ vector clock on the front of the queue. If there’s a vector clock greater than the current, pass the token. If not, remove the timestamp and do the work before passing the token to the next.

Problem 2.

a).

T: U:

write(i,55);

x = read(i);

write(j,66);

write(j,44);

Order of access to i is T, U. Order of access to j is U, T. So the interleaving is not serialized.

b).

T: U:

Lock(i) x = read(i);

lock(j) write(j, 66);

release i

lock(i) write(i,55);

release j

lock(j) write(j,44);

release i,j

3.

a)  
        T         |       U  
    ---------------------------------  
                  |   write(i, 12);  
    x = read(j);  |  
    y = read(i);  |    
                  |   write(k, 66);  
    write(k, 33); |  
                  |   write(l, 22);

It’s serialized, but there’s no two-phase locking schedule for it.

b)

order of access to I is T,U, order of access to k is U,T. not serialized

c)

        T                U  
    ---------------------------------  
   x = read(j);    
                     write(i, 12);  
                     write(k, 66);  
   y = read(i);      
   write(k, 33);   
    write(l, 22);

It’s serialized, but there’s no two-phase locking schedule for it.

d)

        T                U  
    ---------------------------------  
    lock j, x = read(j);    
    lock I, y = read(i);      
    lock k, write(k, 33);   
    commit and release I,j,k

Lock I,write(i, 12);  
     lock j, write(k, 66);  
     lock l, write(l, 22);

Commit and Release I,j,l

Serilized interleaving transactions and a possible strict 2-phase locking schecule is like above. There’s no dirty read in the above schedule.

e)  
  
        T                U  
    ---------------------------------  
                     lock I, write(i, 12);  
    lock j,x = read(j)    
                     lock k, write(k, 66);  
                     lock l, write(l, 22);

release i  
    lock I,y = read(i)     release k  
    lock k,write(k, 33) release l and commit

commit and release j,I,k

This is a serialized interleaving transactions and a possible non-strict 2PL schedule is like above, a dirty read is possible since transaction U might fail to commit and roll back.

4.

a)

possible to occur under pessimistic timestamp concurrency control

b)

not possible

c)

not possible

d)

possible

e)

possible

5.

In order for P1 to R(x)1, the W(x)1 operation in P2 must have finished. If so, there’s no way for P1 to R(x)0, since W(x)0 happens before W(x)1.

This interleaving transactions are not sequentially consistent.

6.

For P3 to R(y)2,P2 much have finished W(y)2. For p2, R(x)1 must have finished. So for has finished W(x)1 and W(x)0 happens before W(x)0. There’s no way for P3 to R(x)0 if it has R(y)2.

So it’s not causally consistent.