Alexandros Karakasidis

Assignment 7

1.

Let *e* and *e'* be concurrent and let *e* occur at P_i and *e'* at P_j. Because the events are concurrent (not related by happened-before) we know that no message sent from P_i at or after event *e* has propagated its timestamp to P_j by the time e' occurs at P_j, and *vice versa*.

The following is true: $Vj[j] \leq Vi[j]$: When a process sends its timestamp vector, the receiver increases its value by one, thus this relationship is valid, if we consider that P_j sends its vector timestamp to P_i.

Using this, $Vj[j] < Vi[j]$ and $Vi[i] < Vj[j]$ so neither, $V(e) \leq V(e')$ nor $V(e') \leq V(e)$.

So, if $V(e)<V(e')$ the two events are not concurrent, which means that they should be with the happens - before relationship and it is trivial that e->e'

2.

9. This is a legal output, if all processes printed out the result, before seeing the write operations of the other process, something which is legal having FIFO consistency, since in FIFO consistency all constraints have to do with writes of single processes.

10. This is a legal signature, if the processes saw the following execution order:

```
x =1;
print(y,z);
z = 1;
print(x,y);
y = 1;
print(x,z);
```

The signature is 001110.

However 001110 cannot be a valid printout. The printout for such an execution will be 001011. No other execution can offer such a printout facilitating sequential consistency, since sequential consistency raises the restriction that all processes should see the same interleaving of operations.

12. The following are the six valid interleavings:

X=1;	x=1;	x=1;	y=1;	y=1;	y=1;
if (y==0);	y=1;	y=1;	if (x==0);	x=1;	x=1;
kill(P2);	if (x==0);	if (y==0);	kill(P1);	if (y==0);	if (x==0);
y=1;	kill(P1);	kill(P2);	x=1;	kill(P2);	kill(P1);
if (x==0);	if (y==0);	if (x==0);	if (y==0);	if (x==0);	if (y==0);
kill(P1);	kill(P2);	kill(P1);	kill(P2);	kill(P1);	kill(P2);

27. An operation can be removed from a queue, when it is known that it has been performed everywhere. This means that the following relationship should be valid:

DEP(W)[k] > VAL(i)[k] for every k and for every local copy Li.

3.

In sequential consistency all processes see the same interleaving of operations, even the order of the operations is not the actual one.

Monotonic Reads

Sequential consistency does not imply monotonic reads, because in monotonic reads, a process always reads the most updated version, something which does not always occur with sequential consistency.

Monotonic Writes

Having monotonic writes, all updates occur with the proper order, and the update happens to the most recent version of the variable. However, nothing is said about the way reads occur. Thus we can assume that processes can follow a sequential consistency scheme, by seeing, all of them, the same interleaving of operations.

Read Your Writes

In this policy, sequential consistency is not implies, since different processes may see writes in a different order, since in the Read Your Writes scheme each process sees its own writes. Thus, different processes see different operations in a different order.

Writes Follow Reads

Neither does this scheme imply sequential consistency, since in *Writes Follow Reads* a process will write on the same or more recent value that has seen, while in sequential consistency there is no assurance about the version of the read variable.