Assignment 6

Georgia Koloniari
Salteas-Kalogeras Panagiotis

1. We do not want to set the clock back to the right time, because some applications use the current clock value to stamp events, on the assumption that clocks always advance.
We use E to refer to the clock that reads 10:27:54.0 (the wrong time) when the real time is 10:27:50, (4 secs before). We adjust our software clock S to tick at rate chosen so that it will be correct after 8 seconds, as follows:
$S = c(E - T_{skew}) + T_{skew}$, where $T_{skew} = 10 : 27 : 54$ and c is to be found.
But $S = T_{skew} + 4$ (the correct time) when $E = T_{skew} + 8$,
So $T_{skew}+4 = c(T_{skew}+8-T_{skew})+T_{skew}$, and c is 0.5.
Finally, $S = 0.5(E-T_{skew})+T_{skew}$ (when $T_{skew} \le E \le T_{skew} + 8$).

2. The client should choose the minimum round-trip time of 20 ms = 0.02 s. It then estimates the current time to be 10:54:28.342 + 0.02/2 = 10:54:28.352. The accuracy is +/- 10 ms.
If the minimum message transfer time is known to be 8 ms, then the setting remains the same but the accuracy improves to +/- 2 ms.

3. Proof by induction
– Basis Case: zero messages exchanged in $e \rightarrow e'$. e and e' are in a single process pj. $e \rightarrow_j e'$ holds. Lamport timestamp rule LC1 implies that $L(e) < L(e')$
– Induction Hypothesis: For up to k messages exchanged, $e \rightarrow e'$ implies $L(e) < L(e')$.
– Induction Step: For k+1 messages exchanged (= k messages and then 1 more message)
• $e \ldots_k f \ldots_1 e'$
• $e \rightarrow f$ (using induction hypothesis) implies $L(e) < L(f)$. Apply rule LC2 to f and e' to get $L(f) < L(e')$. Combining the two, $L(e) < L(e')$.

4. There is a typo and the relationship to be proved is: $V_j[i] \le V_i[i]$.
a) Rule VC2 tells us that p_i is the 'source' of increments to $V_i[i]$, which it makes just before it sends each message; and that p_j increments only as it receives messages containing timestamps with larger entries for p_i, that is when it receives a timestamp T with $T[i] > V_j[i]$. The relationship $V_j[i] \le V_i[i]$ follows immediately.

b) 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*. By the reasoning for (a), it follows that $V_j[i]<V_i[i]$ and $V_i[j]<V_j[j]$ (strict inequalities) and therefore that neither $V(e) \le V(e')$ nor $V(e') \le V(e)$.
Therefore if $V(e)<V(e')$ the two events are not concurrent – they must be related by happened-before. Of the two possibilities, it obviously must be that $e \rightarrow e'$.

5. No, it is sufficient to multicast any other type of message, as long as that message has a timestamp larger than the received message. The condition for delivering a message *m* to the application is that another message has been received from each other process with a large timestamp. This guarantees that there are no more messages underway with a lower timestamp. However, if another message is not multi-casted, we have no such guarantee.

Then, sending an acknowledgement ensures that another message has been received from each other process.

6. The algorithm can be modified as follows: When a process that has started an election itself, receives an ELECTION message without its own id in the list, it compares the id of the initiator of this message with its own id. If its id is bigger then it kills this message, it does not forward it further. Thus, only one ELECTION message is circulating through the ring.

7. If s = synchronization delay and m = minimum time spent in a critical section by any process, then the maximum throughput is $1/(s + m)$ critical-section-entries per second.

8. Two-phase locking in a distributed transaction requires that it cannot acquire a lock at any server after it has released a lock at any server. A client transaction will not request *commit* or *abort* at the coordinator, until after it has made all its requests and had replies from the various servers involved, by which time all the locks will have been acquired. After that, the coordinator sends on the *commit* or *abort* to the other servers which release the locks. Thus all locks are acquired first and then they are all released, which is two-phase locking.