UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Excercise Sheet 7
Solution

Lecture Distributed Systems
Summer Term 2020

Exercise 1: Clock Drift

a) A simple example is the distributed compilation of a program using $make$. Make is controlled by a so-called makefile, in which dependencies e.g. between source and object files are specified, as well as rules specifying how target files can be created from source files. For optimization reasons, such a rule is only applied if the source file has changed since the last time it was used, so that only really necessary compilation processes take place. Make is based on the time stamps of the files stored in the file system that indicate the last change. If the clocks on the individual computers involved deviate from each other, inconsistent states may occur, e.g. a source file may have an older date than the corresponding object file, even though the source code was changed after the last translation process. Erroneously, the recompilation of this file would not take place.

b) With the additional leap seconds inserted, some days take exactly 86401 seconds instead of 86400. This is necessary because an earth rotation does not take exactly $24 * 60 * 60$ seconds, but a small fraction of a second longer. So leap seconds have nothing to do with a possible inaccuracy of the atomic clock, but only serve to adapt to the calendar.

Exercise 2: Clock Synchronisation

There are several ways to synchronize distributed clocks. One of these is Cristian's protocol. A client process P requests the time in a message m_r from a time server S, and receives the time t in a message m_t from S. The problem is that because of the signal propagation time m_t, the time t is out of date again when arriving at P. Therefore, P measures the round trip time T_{round} from sending m_r to receiving m_t and sets its clock to $t + T_{round}/2$ ($T_{round}/2$ is the estimated propagation time of the message m_t).

The accuracy of the procedure is limited by:

- the assumption that the runtimes of m_r and m_t are the same (systematic error).
- the variations in the message durations (statistical error).

In a LAN, the variations (jitter) occurs due to competition for the medium (collisions, buffer times in switches) and due to the processing of messages in the operating systems of P and S. For a LAN, the accuracy is usually within $1ms$. In the Internet, additional inaccuracies occur because the messages are delayed in the routers. For WANs, the accuracy is probably within $5 - 10ms$.

Also with the help of GPS an exact synchronization of clocks is not possible. Due to the known position of satellites and the GPS receiver, the message transfer time can be calculated very accurately, but still varies, e.g., due to weather conditions. In addition, the positions are also afflicted with errors.

Exercise 3: Programming: Clock Synchronization

The programming solution for this task can be found in the archive `l07eFiles.zip`[1] on the lecture's web page.

The clocks on different computers may already be synchronized by the *Network Time Protocol* NTP with sufficient accuracy. Due to the different speed of the local clocks (drift), the synchronization must be repeated regularly.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l07eFiles.zip`

Exercise 4: Clock Synchronization

a) The client should select the request with the minimum round trip time of $20ms$ = $0.02s$. If it can set its clock immediately after the response arrives, it would set its clock to $t_{Server} + T_{round}/2$, i.e. 11:51:25.232 + 0.02/2 = 11:51:25.242. Since the client can usually only determine the answer with the minimum round trip time later (after some requests), it must proceed as follows: Suppose the local time of the client is t. It then sets the clock to $t+(t_{server}+T_{round}/2-t_{arrival}) = t+(t_{server}+T_{round}/2-(t_{send}+T_{round})) = t+25.242-24.734 = t+0.508$, i.e. the clock is advanced by $508ms$.

The accuracy is $\pm 10ms$.

If the minimum transfer time is $8ms$, then the setting remains the same, but the accuracy improves to $\pm 2ms$.

b) To synchronize the clock with an accuracy of $1ms$, if a transmission time of at least $8ms$ is given, the maximum round trip time is $T_{round} = 2 \cdot 1ms + 2 \cdot 8ms = 18ms$.

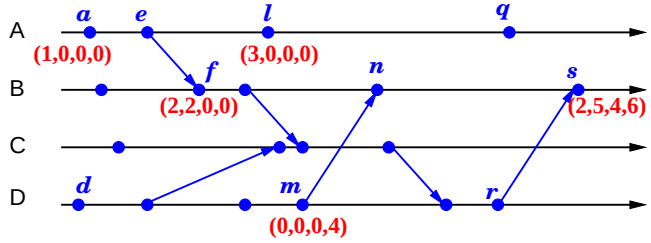
Exercise 5: Lamport and Vector Time

a) Event a is neither before nor after event m in the sense of Lamport's *happened-before* relation. Both events are thus concurrent and cannot have influenced each other causally, since no flow of information can have taken place between these two events.

A (real) causal relationship between two events means that one event caused the other. This can only be safely said in the diagram for the receive events that are triggered by the corresponding send event of the other process.

Whether there really is a causal relationship between two events of the same process cannot be generally determined. This applies, for example, to l and q. Here a causal relationship could exist, but this does not have to be the case (Note: the *happended-before* relation describes **possible** causal relationships, therefore $l \rightarrow q$ applies).

b)



c) m Event e has Lamport time 2, event r has Lamport time 8. From this it can **not** be concluded that $e \rightarrow r$ applies (although this is the case here "by chance"). From the fact that the time stamp of r is not smaller than that of e, it can be concluded that $r \not\rightarrow e$ applies.

Exercise 6: *Happened-Before* Relation, Lamport and Vector Time

a) **No** statement can be made about the *happened-before* relation between events (in different processes) based on Lamport timestamps. The same applies to real time.

In this very special case, assuming that the given events (a_1, a_2, a_3, a_4, b_1, b_2, c_1, c_2) represent all occurred events, the following can be concluded: Since the Lamport time jumped to 5 at event c_2, this can only have happened by receiving a message with Lamport time 4. The only event with Lamport time 4 is a_4, so a_4 must have been the corresponding send event. This means that $a_4 \rightarrow c_2$ must apply.

b) $V(a_1) = (1,0,0) \not< (0,1,0) = V(b_1)$ and $V(b_1) = (0,1,0) \not< (1,0,0) = V(a_1)$, i.e. $a_1 \not\rightarrow b_1$ and $b_1 \not\rightarrow a_1$, i.e. a_1 and b_1 are concurrent. Their order in real time can be arbitrary.

$V(c_1) = (0,0,1) < (0,2,1) = V(b_2)$, i.e. $c_1 \rightarrow b_2$. c_1 must therefore have taken place in real time before b_2.

$V(a_4) = (4,1,0) < (4,1,2) = V(c_2)$, i.e. $a_4 \rightarrow c_2$. a_4 must therefore have taken place in real time before c_2.

c) For the vector time: $V(a) < V(b) \Leftrightarrow a \rightarrow b$.
For the Lamport time only $a \rightarrow b \Rightarrow L(a) < L(b)$ applies, but not vice versa.

Exercise 7: Programming: Vector Clock

You will find the solution to this problem in in the archive `l07eFiles.zip`[2] on the lecture's web page.

[2] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l07eFiles.zip`