UNIVERSITÄT SIEGEN

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

Aufgabenblatt 7
Musterlösung

Vorlesung Verteilte Systeme
Sommersemester 2020

Aufgabe 1: Uhrendrift

a) Ein einfaches Beispiel ist das verteilte Kompilieren eines Programms mittels `make`. Make wird über ein sogenanntes Makefile gesteuert, in dem Abhängigkeiten z.B. zwischen Quell- und Objektdateien sowie Regeln, die angeben, wie Zieldateien aus Quelldateien erzeugt werden können, spezifiert sind. Aus Optimierungsgründen wird eine solche Regel nur angewendet, wenn sich seit der letzten Anwendung die Quelldatei verändert hat, damit eben nur wirklich notwendige Kompiliervorgänge stattfinden. Dabei orientiert sich Make an den im Dateisystem abgelegten Zeitstempeln der Dateien, die die letzte Änderung angeben. Weichen nun die Uhren auf den einzelnen beteiligten Rechnern voneinander ab, so kann es zu inkonsistenten Zuständen kommen, z.B. dazu, dass eine Quelldatei ein älteres Datum als die zugehörige Objektdatei aufweist, obwohl der Quellcode nach dem letzten Übersetzungsvorgang verändert wurde. Fehlerhafterweise würde dann die erneute Kompilierung dieser Datei ausbleiben.

b) Mit den zusätzlich eingefügten Schaltsekunden wird erreicht, dass manche Tage statt 86400 genau 86401 Sekunden dauern. Notwendig ist dies, da eine Erddrehung eben nicht exakt $24 * 60 * 60$ Sekunden dauert, sondern einen kleinen Sekundenbruchteil länger. Schaltsekunden haben also nichts mit einer evtl. Ungenauigkeit der Atomuhr zu tun, sondern dienen lediglich einer Anpassung an den Kalender.

Aufgabe 2: Uhrensynchronisation

Es gibt mehrere Möglichkeiten zur Synchronisierung verteilter Uhren. Eine von diesen ist Cristians Protokoll. Ein Client-Prozess P fordert die Zeit in einer Nachricht m_r von einem Zeit-Server S an, und empfängt den Zeitwert t in einer Nachricht m_t von S. Das Problem ist, daß wegen der Signallaufzeit von m_t die Zeit t bis zur Ankunft bei P schon wieder veraltet ist. P mißt daher die Round-Trip-Zeit T_{round} vom Versenden von m_r bis zum Empfang von m_t und setzt seine Uhr auf $t + T_{round}/2$ ($T_{round}/2$ ist die geschätzte Laufzeit der Nachricht m_t).

Die Genauigkeit des Verfahrens wird eingeschränkt durch:

- die Annahme, daß die Laufzeiten von m_r und m_t gleich sind (systematischer Fehler).
- die Streuungen in den Nachrichtenlaufzeiten (statistischer Fehler).

In einem LAN entstehen Streuungen durch den Wettbewerb um das Medium (Kollisionen, Pufferzeiten in Switches) und durch die Verarbeitung der Nachrichten im Betriebssystem von P und S. Für ein LAN ist die Genauigkeit vermutlich innerhalb $1ms$. Im Internet entstehen zusätzliche Ungenauigkeiten, weil die Nachrichten in den Routern verzögert werden. Für WANs ist die Genauigkeit vermutlich innerhalb $5 - 10ms$.

Auch mit Hilfe von GPS ist eine exakte Synchronisierung von Uhren nicht möglich. Wegen der bekannten Position von Satellit und GPS-Empfänger kann die Nachrichtenlaufzeit zwar sehr genau berechnet werden, schwankt aber z.B. aufgrund von Wetterbedigungen. Zudem sind auch die Positionen mit Fehlern behaftet.

Aufgabe 3: Programmierung: Uhrensynchronisation

Die Programmierlösung dieser Aufgabe finden Sie im Archiv `l07Files.zip`[1] auf der Vorlesungswebseite.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l07Files.zip`

Die Uhren auf verschiedenen Rechnern werden u.U. bereits durch das *Network Time Protocol* NTP hinreichend genau synchronisiert. Durch den unterschiedlich schnellen Gang der lokalen Uhren (Drift) muss die Synchronisation regelmäßig wiederholt werden.

Aufgabe 4: Uhrensynchronisation

a) Der Client sollte die Anfrage mit der minimalen Round-Trip-Zeit von $20ms$ wählen = $0.02s$. Falls er seine Uhr unmittelbar nach Ankunft der Antwort setzen kann, würde er seine Uhr auf $t_{Server} + T_{round}/2$, also auf 11:51:25.232 + 0.02/2 = 11:51:25.242 Uhr setzen. Da der Client in der Regel aber erst später (nach einigen Anfragen) diejenige mit der minimalen Round-Trip-Zeit bestimmen kann, muß er wie folgt vorgehen: Angenommen, die lokale Zeit des Clients ist t. Er setzt dann die Uhr auf $t + (t_{Server} + T_{round}/2 - t_{Ankunft})$ = $t + (t_{Server} + T_{round}/2 - (t_{Absenden} + T_{round}))$ = $t + 25.242 - 24.734 = t + 0.508$, d.h., die Uhr wird um $508ms$ vorgestellt.

Die Genauigkeit ist $\pm 10ms$.

Wenn die minimale Übertragungszeit $8ms$ beträgt, dann bleibt die Einstellung die selbe, aber die Genauigkeit verbessert sich auf $\pm 2ms$.

b) Um die Uhr innerhalb von $1ms$ zu synchronisieren, wenn eine Übertragungszeit von mindestens $8ms$ gegeben sei, ist die maximale Round-Trip-Zeit $T_{round} = 2 \cdot 1ms + 2 \cdot 8ms = 18ms$.

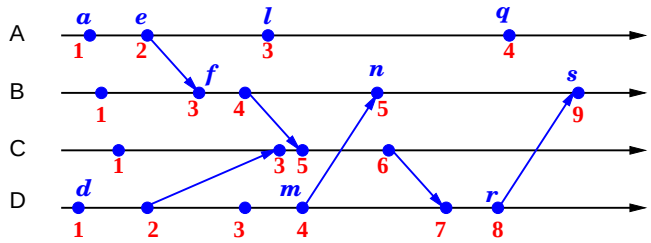
Aufgabe 5: Lamport- und Vektorzeit

a) Ereignis a ist im Sinne der Lamport'schen *Happened-Before-Relation* weder vor noch nach Ereignis m. Beide Ereignisse sind also nebenläufig und können sich kausal nicht beeinflusst haben, da kein Informationsfluss zwischen diesen beiden Ereignissen stattgefunden haben kann.

Ein (wirklicher) kausaler Zusammenhang zwischen zwei Ereignissen bedeutet, dass ein Ereignis das andere verursacht hat. Das kann im Diagramm nur bei den Empfangsereignissen sicher gesagt werden, die vom entsprechenden Sendeereignis des anderen Prozesses ausgelöst werden.

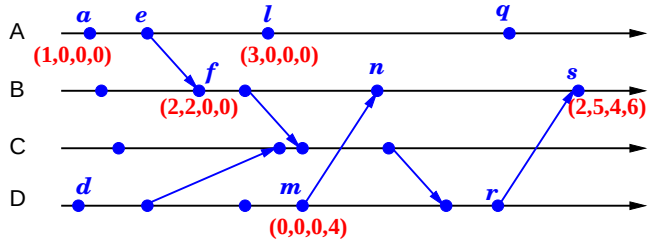
Ob zwischen zwei Ereignissen desselben Prozesses wirklich ein kausaler Zusammenhang besteht, kann nicht allgemein festgestellt werden. Dies gilt z.B. für l und q. Hier könnte ein kausaler Zusammenhang bestehen, das muß aber nicht so sein (Anmerkung: die *happended-before*-Relation beschreibt **mögliche** kausale Zusammenhänge, daher gilt $l \rightarrow q$).

b)



c) Ereignis e hat Lamport-Zeit 2, Ereignis r die Lamport-Zeit 8. Daraus kann **nicht** gefolgert werden, daß $e \rightarrow r$ gilt (obwohl das hier „zufällig" der Fall ist). Aus der Tatsache, daß der Zeitstempel von r aber nicht kleiner ist als der von e, kann gefolgert werden, daß $r \not\rightarrow e$ gilt.

d)



e) $V(l) = (3,0,0,0) \not< (0,0,0,4) = V(m)$ und $V(m) = (0,0,0,4) \not< (3,0,0,0) = V(l)$, d.h. $l \not\rightarrow m$ und $m \not\rightarrow l$, d.h. l und m sind nebenläufig.

$V(f) = (2,2,0,0) < (2,5,4,6) = V(s)$ d.h. $f \rightarrow s$.

Aufgabe 6: *Happened-Before*-Relation, Lamport- und Vektorzeit

a) Über die *happened-before* Relation zwischen Ereignissen (in verschiedenen Prozessen) kann aufgrund von Lamport-Zeitstempeln **keine** Aussage gemacht werden. Gleiches gilt für die reale Zeit.

In diesem ganz speziellen Fall kann unter der Annahme, daß die gegebenen Ereignisse (a_1, a_2, a_3, a_4, b_1, b_2, c_1, c_2) alle aufgetretenen Ereignisse darstellen, noch folgendes schließen: Da die Lamport-Zeit bei Ereignis c_2 auf 5 gesprungen ist, kann das nur durch den Empfang einer Nachricht mit Lamport-Zeit 4 passiert sein. Das einzige Ereignis mit Lamport-Zeit 4 ist a_4, damit muß a_4 das zugehörige Sendeereignis gewesen sein. D.h. es muß $a_4 \rightarrow c_2$ gelten.

b) $V(a_1) = (1,0,0) \not< (0,1,0) = V(b_1)$ und $V(b_1) = (0,1,0) \not< (1,0,0) = V(a_1)$, d.h. $a_1 \not\rightarrow b_1$ und $b_1 \not\rightarrow a_1$, d.h. a_1 und b_1 sind nebenläufig. Ihre Reihenfolge in realer Zeit kann beliebig sein.

$V(c_1) = (0,0,1) < (0,2,1) = V(b_2)$, d.h. $c_1 \rightarrow b_2$. c_1 muß demnach auch in realer Zeit vor b_2 stattgefunden haben.

$V(a_4) = (4,1,0) < (4,1,2) = V(c_2)$, d.h. $a_4 \rightarrow c_2$. a_4 muß demnach auch in realer Zeit vor c_2 stattgefunden haben.

c) Für die Vektorzeit gilt: $V(a) < V(b) \Leftrightarrow a \rightarrow b$.
Für die Lamport-Zeit gilt nur $a \rightarrow b \Rightarrow L(a) < L(b)$, aber nicht umgekehrt.

Aufgabe 7: Programmierung: Vektor-Uhr

Die Lösung dieser Aufgabe finden Sie im Archiv `l07Files.zip`[2] auf der Vorlesungswebseite.

[2] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l07Files.zip`