UNIVERSITÄT SIEGEN

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

Aufgabenblatt 9
Musterlösung

Vorlesung Verteilte Systeme
Sommersemester 2020

Aufgabe 1: Zentralisierter Algorithmus zum wechselseitigen Ausschluß

a) Anforderungen könnten abhängig von ihrer Bedeutung Prioritätsstufen zugeorgnet werden. Der Koordinator könnte dann die Anforderungen mit der höchsten Priorität zuerst bedienen.

b) Angenommen, der Algorithmus gibt vor, dass jede Anforderung unmittelbar beantwortet wird, entweder durch eine Erlaubnis oder durch eine Verweigerung. Befinden sich keine Prozesse in kritischen Bereichen und in den Warteschlangen, ist ein Absturz nicht fatal.

Der nächste Prozess, der eine Berechtigung anfordert, erhält keine Antwort und kann dann die Wahl eines neuen Koordinators initiieren. Das System kann noch robuster gemacht werden, indem der Koordinator jede eingehende Anforderung auf der Festplatte speichert, bevor er eine Antwort zurücksendet. Auf diese Weise kann im Falle eines Absturzes der neue Koordinator die Liste der aktiven kritischen Bereiche und die Warteschlange rekonstruieren, indem er die Datei von der Festplatte liest.

Aufgabe 2: Algorithmus von Ricart und Agrawala

Angenommen, ein Prozess verweigert die Berechtigung und stürzt dann ab. Der anfordernde Prozess glaubt, er sei noch am Leben, aber die Berechtigung trifft niemals ein. Ein Ausweg ist, den Anforderer zu veranlassen, dass er nicht nur blockiert, sondern dass er eine bestimmte Zeidauer lang *schläft*, wonach er alle Prozesse abfragt, die die Berechtigung verweigert haben, um zu prüfen, ob sie noch laufen.

Aufgabe 3: Deadlocks - Ricart und Agrawala

Das ist von den Grundregeln abhängig. Wenn die Prozesse streng sequentiell in kritische Bereiche eintreten, d.h. ein Prozess in einem kritischen Bereich darf nicht versuchen, in einen weiteren kritischen Bereich einzutreten, gibt es keine Möglichkeit, dass er blockiert, während er eine Ressource (d.h. einen kritischen Bereich) hält, die ein anderer Prozess braucht. Das System ist Deadlock-frei. Kann andererseits Prozess 0 in den kritischen Bereich A eintreten und dann versuchen, in den kritischen Bereich B einzutreten, kann ein Deadlock auftreten, wenn ein anderer Prozess versucht, die kritischen Bereiche in umgekehrter Reihenfolge zu betreten. Der Algorithmus von Ricart und Agrawala selbst trägt nicht zu dem Deadlock bei, weil jeder kritische Bereich unabhängig von den anderen behandelt wird.

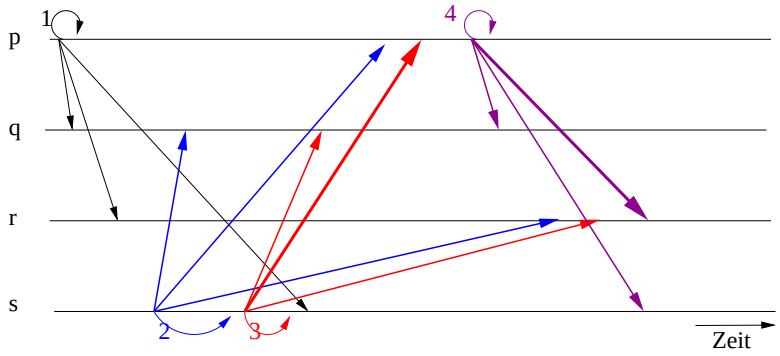
Aufgabe 4: Programmierung: Ricart/Agrawala

Die Lösung dieser Aufgabe finden Sie im Archiv `l09Files.zip`[1] auf der Vorlesungswebseite.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l09Files.zip`

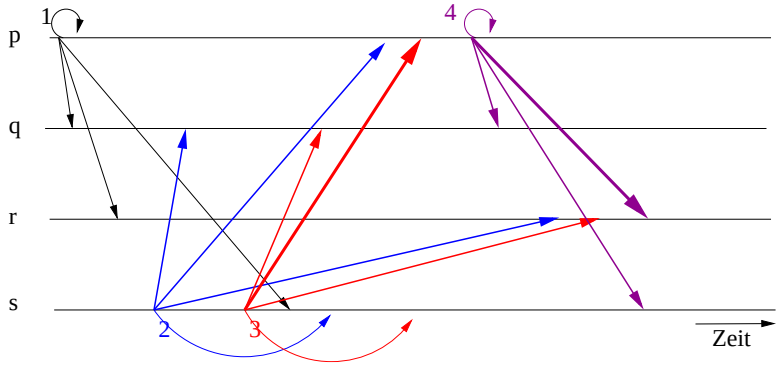
Aufgabe 5: Reihenfolgegarantien beim Multicast

a) Bei einem kausal sortierten Multicast muss die Nachricht 3 in allen Prozessen (insbes. Prozeß p) nach der Nachricht 2 eintreffen (da 3 kausal von 2 abhängt). Ebenso muß Nachricht 4 in allen Prozessen (insbes. Prozeß r) nach Nachricht 3 eintreffen.



Damit ist aber noch keine vollständige Sortierung gewährleistet (z.B. Prozeß p: Reihenfolge 1,2,3,4; Prozeß s: Reihenfolge 2,3,1,4).

Eine vollständige Sortierung wäre gegeben, wenn alle Nachrichten von allen Prozessen in derselben Reihenfolge ausgeliefert werden.



b) Damit der Benutzer alle Veröffentlichungen erhält, wird ein **zuverlässiger** Multicast benötigt.

Eine **FIFO-Reihenfolge** wird benötigt, damit die Veröffentlichungen eines Benutzer, z.B. A. Bauer, überall in derselben Reihenfolge empfangen werden. Die Benutzer können dann konsistent über die „zweite Veröffentlichung“ von A. Bauer diskutieren. Zudem könnte sich der zweite Beitrag ja auf den ersten beziehen.

Eine **kausale-Reihenfolge** wird ebenfalls benötigt, weil die Nachrichten, deren Themen mit *Re:* anfangen, nach den Nachrichten erscheinen sollten, auf die sie sich beziehen. Andernfalls könnte eine Verzögerung einer Nachricht dazu führen, dass z.B. die Nachricht *Re: RPC Prinzip* vor der ursprünglichen Nachricht *RPC Prinzip* erscheint.

Wäre die Multicast-Auslieferung vollständig sortiert, wäre die Nummerierung in der linken Spalte konsistent zwischen den Benutzern. Dies ist aber nicht zwingend notwendig.

In der Praxis implementiert das USENET-System, das ein solches schwarzes Brett nachbildet, weder eine kausale noch eine vollständige Reihenfolge. Der dazu notwendige Kommunikationsaufwand würde ihre Vorteile bei weitem überwiegen.

Aufgabe 6: Kausal sortierter Multicast

a) Kausale Sortierung heißt wenn eine Nachrich b kausal von einer Nachricht a abhängt (bzw. abhängen könnte), dann empfangen alle Prozesse a vor b.

Der Multicast ist kausal sortiert, weil:

- jede Nachricht mit dem Lamport-Zeitstempel des Sende-Ereignisses versehen wird, (der nach jeder Nachricht inkrementiert wird)
- und die Empfänger ankommende Nachrichten in der Reihenfolge der Lamport-Zeitstempel ausliefern.

Wenn also $a \rightarrow b$ gilt, dann auch $L(a) < L(b)$, d.h., b wird nach a an den Empfänger ausgeliefert (obwohl b natürlich vor a am Empfängerknoten ankommen könnte; b wird dann „einfach“ zurückgehalten).

b) Wie oben angedeutet, darf der Empfängerknoten eine Nachricht b erst dann ausliefern, wenn alle Nachrichten mit kleinerem Zeitstempel als $L(b)$ bereits ausgeliefert wurden. Woher aber soll der Empfängerknoten wissen, daß keine Nachricht m mit $L(m) < L(b)$ mehr unterwegs ist?

Eine einfache Lösung wäre, ankommende Nachrichten mit kleinerem Zeitstempel als dem der zuletzt ausgelieferten Nachricht einfach zu verwerfen. Der Multicast wäre dann zwar noch kausal geordnet, aber nicht mehr zuverlässig.

Eine andere Lösung ist denkbar, wenn die einzelnen Kanäle FIFO-Semantik besitzen: wenn ein Knoten von allen anderen Knoten Nachrichten mit Zeitstempel größer als $L(b)$ erhalten hat, kann er b ausliefern, da sicher keine Nachricht m mit $L(m) < L(b)$ mehr kommen kann. Als Problem bleibt, daß auf Nachrichten aller anderen Prozesse gewartet werden muß (möglicherweise sehr lange, wenn ein Knoten gerade nichts zu senden hat). Mögliche Lösungen wären:

- Versenden periodischer Broadcasts (Problem: immer noch lange Verzögerung, Overhead)
- Versenden von Empfangs-Bestätigungen per Multicast (Problem: Overhead)

Eine effiziente Lösung ist mit Vektoruhren möglich. Die Vektoruhren zählen dabei nur die Multicast-Ereignisse. Der Empfängerknoten weiß dann aus dem Vektor-Zeitstempel einer eintreffenden Nachricht genau, auf welche Nachrichten er noch warten muss (weil das Element i des Vektor-Zeitstempels genau angibt, wieviele Ereignisse in Prozeß i aufgetreten sind, die kausal vor dem Versenden der Nachricht liegen).

c) Das Problem ist, daß zwei Nachrichten denselben Lamport-Zeitstempel haben könnten. Die Empfänger wüßten dann nicht eindeutig, welche Nachricht zuerst ausgeliefert werden soll. Zwei Empfänger könnten die Nachrichten also in unterschiedlicher Reihenfolge ausliefern.

Also Abhilfe müsste man eine lineare Ordnung der Nachrichten realisieren, z.B. indem man die Lamport-Zeit noch mit einer Prozeß-ID des Senders kombiniert (vgl. Ricart-Agrawala-Alogrithmus).