

Aufgabenblatt 9

(Zu bearbeiten bis 19.12.)

Vorlesung Verteilte Systeme Wintersemester 2024/25

Aufgabe 1: Algorithmus von Ricart und Agrawala

Der Algorithmus von Ricart und Agrawala hat das Problem, dass falls ein Prozess abgestürzt ist und nicht auf eine Anforderung von einem anderen Prozess reagiert, in einem kritischen Bereich einzutreten, die fehlende Antwort als Verweigerung der Berechtigung interpretiert wird. Eine Methode, um abgestürzte Prozesse einfach zu erkennen, besteht darin, alle Anforderungen unmittelbar (ggf. negativ) zu beantworten. Gibt es Situationen, wo selbst diese Methode nicht ausreichend ist?

Aufgabe 2: Deadlocks - Ricart und Agrawala

Ein verteiltes System kann mehrere voneinander unabhängige kritische Bereiche haben. Angenommen, Prozess 0 will in den kritischen Bereich A und Prozess 1 will in den kritischen Bereich B eintreten. Kann der Algorithmus von Ricart und Agrawala zu Deadlocks führen? Erklären Sie Ihre Antwort.

Aufgabe 3: Programmierung: Ricart/Agrawala (**Pflichtaufgabe für 6 LP, Abgabe über moodle!**)

Im Archiv [u09Files.zip](#)¹ auf der Vorlesungswebseite finden Sie die Realisierung eines Prozeß-Systems mit mehreren Server-Prozessen. Die Server erhalten zu Beginn vom Client über eine Nachricht genau eine Aufgabe, die sie lokal bearbeiten. Zur Bearbeitung müssen sich die Server in einen kritischen Abschnitt begeben, in dem eine Datei (`ressource.txt`) geschrieben wird.

Realisieren Sie auf Basis des gegebenen Codes den Algorithmus zum wechselseitigen Ausschluss nach Ricart und Agrawala, so daß zu jeder Zeit immer nur ein Server in die Datei schreiben kann. Vervollständigen Sie dazu die Methoden `lockResource()`, `unlockResource()` und `handleMutexMessage()` in `Server.java` und ergänzen Sie ggf. notwendige Deklarationen und Initialisierungen. Zur Vereinfachung dürfen Sie annehmen, daß ein Server nur genau einmal einen Auftrag vom Client bekommt, während der Bearbeitung des Auftrags also kein neuer Auftrag eintreffen kann.

Die für den Algorithmus notwendige Realisierung von Lamport-Zeitstempeln ist bereits vollständig vorgegeben. Der Absender einer Nachricht kann ebenfalls ermittelt werden (Methode `getSender()` der Klasse `Message`).

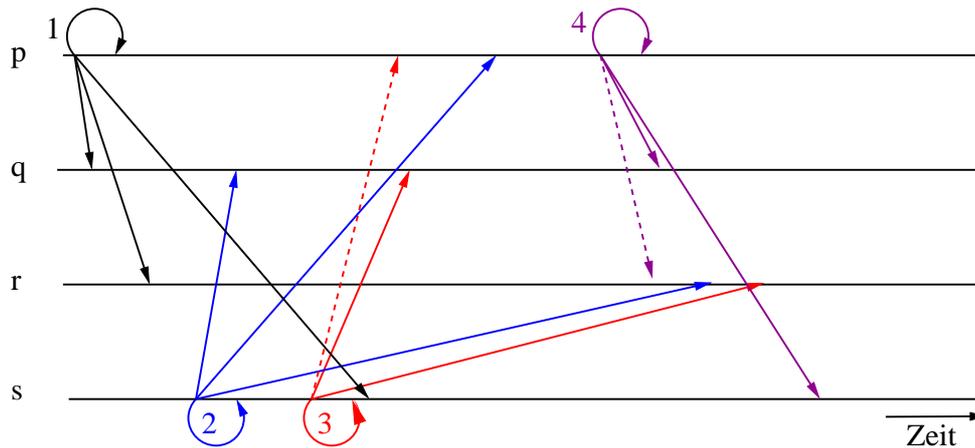
Testen Sie Ihre Lösung mit unterschiedlich vielen, gleichzeitigen Aufträgen und unterschiedlich vielen Servern. Die Zahl der Aufträge kann beim Client als Kommandozeilen-Argument angegeben werden (maximal so viele, wie Server vorhanden sind).

Aufgabe 4: Reihenfolgegarantien beim Multicast

In der Vorlesung wurden verschiedene Varianten für Reihenfolge-Garantien beim Multicast vorgestellt (s. Kap. 7.3).

- a) In der Abbildung sind 4 Prozesse (p, q, r, s) gezeigt und vier Multicast-Nachrichten (1, 2, 3, 4).

¹<http://www.bs.informatik.uni-siegen.de/web/wismueller/v1/vs/u09Files.zip>



Nachricht 4 ist im Beispiel kausal abhängig von Nachricht 3.

Wann muß die Auslieferung der Nachrichten 3 und 4 stattfinden, so dass eine FIFO und eine kausale Sortierung für alle Nachrichten gewährleistet wird? Hinweis: Sie müssen nur die gestrichelten Pfeile richtig zeichnen. Ist dann auch eine vollständige Sortierung gewährleistet?

- b) Um die Multicast-Sortierung zu verdeutlichen, betrachten wir die Anwendung „elektronisches schwarzes Brett“, bei der die Benutzer Nachrichten auf einem schwarzen Brett hinterlassen. Jeder Benutzer führt einen Applikationsprozess für das schwarze Brett aus. Es gibt mehrere Diskussionsthemen, zu denen jeweils eine eigene Prozeßgruppe gehört. Der Prozess eines Benutzers ist Mitglied in der Gruppe für das Thema, an dem dieser interessiert ist, so dass der Benutzer nur Nachrichten zu dem jeweiligen Thema erhält. Wenn ein Benutzer eine Nachricht auf einem schwarzen Brett anbringt, multicastet die Applikation diese an die entsprechende Gruppe.

Ein Benutzer X könnte dann folgende Anzeige aus dem Programm für das schwarze Brett erhalten:

Schwarzes Brett zum Thema: Kommunikation in verteilten Systemen

Nachricht	von	Titel der Nachricht
27	A. Bauer	UDP & TCP Kommunikation
28	C. Fritz	RPC Prinzip
29	G. Meier	Re: UDP & TCP Kommunikation
30	K. Wim	RPC 1
31	A. Bauer	Re: RPC Prinzip
32	K. Wim	RPC 2

Beachten Sie, dass die Nachrichten, deren Themen mit *Re:* anfangen, nach den Nachrichten erscheinen, auf die sie sich beziehen.

Welche sortierte Multicast-Verfahren werden in der Applikation gebraucht und wo genau? Wenn der Multicast vollständig sortiert wäre, was kann man dann über die Nummerierung in der linken Spalte sagen?

Aufgabe 5: Vollständig sortierter Multicast

Um vollständig sortierten Multicast mit der Hilfe eines Sequenzers zu implementieren, ist ein Ansatz, die Nachricht zuerst an den Sequenzer weiterzugeben, der ihr eine eindeutige Nummer zuweist und sie anschließend per Multicast weitergibt. Nennen Sie einen alternativen Ansatz und vergleichen Sie die zwei Lösungen.

Aufgabe 6: Kausal sortierter Multicast (Pflichtaufgabe für 6 LP, Abgabe über moodle!)

Angenommen, ein System stellt zuverlässigen Multicast mit FIFO-Sortierung zur Verfügung. Man könnte auf die Idee kommen, darauf aufbauend einen kausal sortierten Multicast wie folgt zu implementieren:

- jede Nachricht wird mit dem Lamport-Zeitstempel des Sende-Ereignisses versehen,

- die Empfänger-Knoten liefern ankommende Nachrichten in einer Reihenfolge an die Empfängerprozesse aus, die konsistent mit der Ordnung der Zeitstempel ist.
- a) Begründen Sie, warum dieser Multicast kausal sortiert ist.
 - b) Wo ist das Problem bei dieser Implementierung? (**Tip**: wann kann eine Nachricht an den Empfängerprozess ausgeliefert werden?) Wie könnte es gelöst werden?
 - c) Begründen Sie, warum dieser Multicast nicht vollständig sortiert ist. Wie könnte man eine vollständige Sortierung erreichen?