UNIVERSITÄT SIEGEN

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

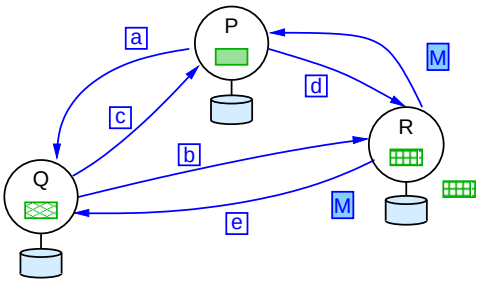
Aufgabenblatt 8
Musterlösung

Vorlesung Verteilte Systeme
Sommersemester 2020

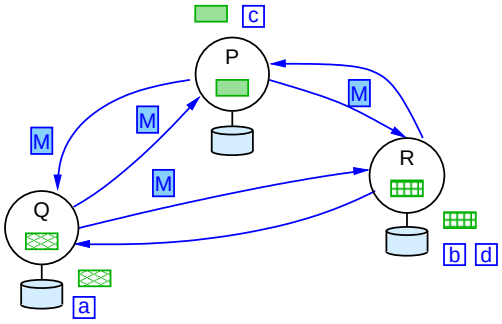
Aufgabe 1: Schnappschuß-Algorithmus nach Chandy/Lamport

a)

Anfang des Algorithmus
(R initiiert den Schnappschuß)



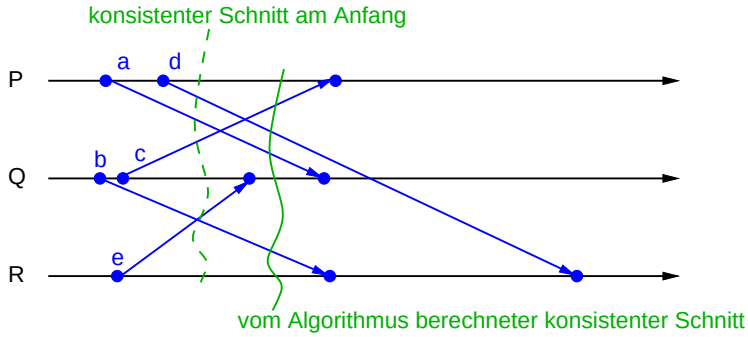
Ende des Algorithmus
(R und Q empfangen die letzten Marker-Nachrichten)



Ablauf des Algorithmus:

1. Process R initiiert den Schnappschuß, sichert seinen lokalen Zustand und sendet Marker (M) an P und Q;
2. Process P empfängt den Marker von R, sichert Zustand und sendet Marker an R und Q;
3. Process Q empfängt und verarbeitet die Nachricht e;
4. Process Q empfängt den Marker von R, sichert seinen lokalen Zustand und sendet Marker an R und P;
5. P, Q, R sichern die eintreffende Nachrichten (c, a sowie b und d), bis alle Marker empfangen sind.

b) Der Schnitt besteht aus den lokalen Zuständen von P,Q und R und den Nachrichten a, b, c, d.



Aufgabe 2: Schnappschuß-Algorithmus nach Chandy/Lamport

1. P sendet m.
2. P initiiert den Schnappschuss, speichert den Zustand 101, und sendet einen Marker an Q.
3. Q empfängt m, sein Zustand ist damit 102.

4. Q empfängt den Marker und speichert seinen Zustand 102. Damit hat Q bereits alle Marker empfangen, der aufgezeichnete Zustand des Nachrichten-Kanals von P nach Q ist damit leer.
5. Q sendet einen Marker an P
6. Q sendet m wieder zurück an P
7. P empfängt den Marker
8. P speichert den den Zustand des Nachrichten-Kanals von Q nach P. Dieser ist leer, da P vor dem Marker keine Nachricht empfangen hat.

Ermittelter Zustand: $P : 101, Q : 102$, beide Kanäle sind leer.

Anmerkung: Q könnte Schritt 6 auch unmittelbar nach Schritt 3 ausführen. In diesem Fall wäre der Zustand: $P : 101, Q : 102$, Kanal $Q \mapsto P$ enthält Nachricht m.

Aufgabe 3: Programmierung: Schnappschuß-Algorithmus nach Chandy/Lamport

Die Lösung dieser Aufgabe finden Sie im Archiv `l08Files.zip`[1] auf der Vorlesungswebseite.

Aufgabe 4: Bully-Algorithmus

a) Alle höher nummerierten Prozesse erhalten zwei ELECTION-Nachrichten. Sie beantworten auch beide mit OK, so dass beide Initiatoren die Wahl abbrechen. Wenn ein Prozess eine ELECTION-Nachricht erhält, aber bereits eine Wahl abhält, so ignoriert er diese (redundante) Nachricht. Die höher nummerierten Prozesse ignorieren also jeweils die zweite ELECTION-Nachricht und führen Ihre Wahl wie üblich durch.

b) Zunächst ist zu beachten, dass dies ein unerwünschtes Verhalten ist, wenn es keinen direkten Vorteil bei der Verwendung eines höher nummerierten Prozesses gibt, denn eine Wiederwahl ist meistens verschwenderisch. Jedoch kann die Nummerierung von Prozessen ihren relativen Nutzen wiederspiegeln (beispielsweise höher nummerierte Prozesse, die auf schnelleren Maschinen ausgeführt werden). In diesem Fall kann der Nutzen, der durch eine Wiederwahl entsteht, die Kosten dieser wert sein. Die Kosten für eine Wiederwahl umfassen auch die erneute Kommunikation, also die Runden, in denen die ELECTION-Nachrichten versendet werden, sowie die Übermittlung von Statusnachrichten von Koordinator zu Koordinator.

Um eine Wiederwahl zu vermeiden, könnte ein sich erholender Prozess lediglich eine requestStatus-Nachricht an die nächsten aufeinanderfolgenden unteren nummerierten Prozesse senden. Damit könnte er herausfinden, ob ein anderer Prozess bereits gewählt wurde und wählt sich selber nur dann, wenn er eine negative Antwort erhält. Danach kann der Algorithmus wie zuvor arbeiten: Wenn der gerade erholte Prozess entdeckt, dass der Koordinator versagt oder er eine ELECTION-Nachricht empfängt, sendet er eine Koordinator-Nachricht an die verbleibenden Prozesse.

c) Mit denen in der Frage genannten Voraussetzungen können wir nicht garantieren, dass jederzeit ein eindeutiger Prozess gewählt werden kann.

Stattdessen können wir es als ausreichend akzeptieren, Untergruppen von Prozessen zu bilden, die ihrem Koordinator als solchem Zustimmen und zu erlauben, dass mehrere solcher Untergruppen auf einmal existieren.

Wenn zum Beispiel ein Netzwerk in zwei geteilt wird, könnten wir zwei grösstmögliche Untergruppen bilden, von denen jede den Prozess mit der höchsten Kennung unter seinen Mitgliedern zum Koordinator wählt. Allerdings sollte, für den Fall, dass die Partitionen wieder zusammengeführt werden, sichergestellt werden, dass aus den beiden Gruppen eine Einzige mit nur einem Koordinator entsteht.

Der sog. Invitation-Algorithmus von Hector Garcua-Molina erreicht dies. Er wählt einen einzigen Koordinator innerhalb einer Untergruppe deren Mitglieder miteinander kommunizieren können. In regelmässigen Abständen fragt der Koordinator andere Mitglieder der gesamten Reihe von Prozessen an, um zu versuchen sich mit anderen Gruppen zu verschmelzen. Wenn ein anderer Koordinator gefunden wird, sendet ein Koordinator eine Einladungs-Nachricht, um ihn dazu einzuladen, eine zusammengeführte Gruppe zu bilden. Wenn ein Prozess eine Nichterreichbarkeit oder einen Misserfolg seines Koordinators vermutet, führt er, wie im Bully-Algorithmus, eine Wahl durch.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l08Files.zip`

Aufgabe 5: Ring-Algorithmus

Wenn ein Prozess eine ELECTION-Nachricht empfängt, prüft er, wer sie gesendet hat. Hat er sie selbst gesendet (d.h. seine Nummer befindet sich am Listenanfang), wandelt er die Nachricht in eine COORDINATOR- Nachricht um, wie in der Vorlesung beschrieben. Hat er keine ELECTION- Nachricht gesendet, trägt er seine Prozessnummer ein und gibt die Nachricht im Ring weiter.

Hat er jedoch zuvor seine eigene ELECTION-Nachricht gesendet und entdeckt soeben einen Konkurrenten, vergleicht er die Prozessnummer des Ursprungsprozesses mit seiner eigenen. Hat der andere Prozess eine niedrigere Nummer, verwirft er die Nachricht, statt sie weiterzugeben.

Hat der andere Prozess eine höhere Nummer, wird die Nachricht wie üblich weitergegeben. Auf diese Weise bleibt nur die ELECTION-Nachricht mit der höheren Nummer übrig, wenn mehrere ELECTION-Nachrichten gesendet werden. Die restlichen werden entlang des Weges verworfen.