UNIVERSITÄT SIEGEN

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

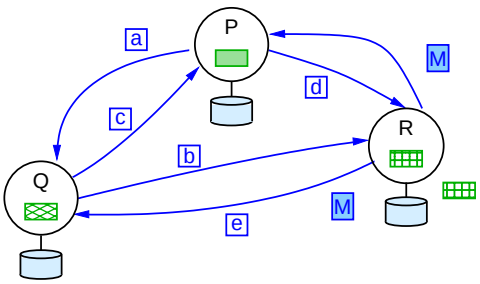
Excercise Sheet 8
Solution

Lecture Distributed Systems
Summer Term 2021

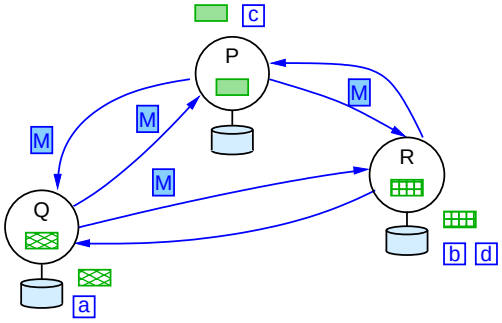
Exercise 1: Snapshot Algorithm According to Chandy/Lamport

a)

Start of algorithm
(R initiates the snapshot)



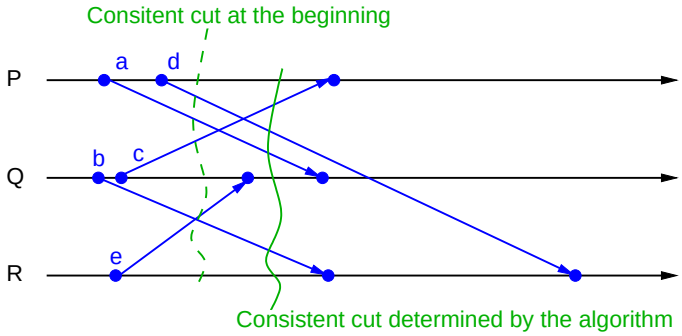
End of algorithm
(R and Q receive last marker message)



Flow of the algorithm:

1. Process R initiates the snapshot, saves its local state and sends markers (M) to P and Q;
2. Process P receives the marker from R, saves the state and sends markers to R and Q;
3. Process Q receives and processes the message e;
4. Process Q receives the marker from R, saves its local state and sends markers to R and P;
5. P, Q, R save the incoming messages (c, a as well as b and d) until all markers are received.

b) The cut consists of the local states of P, Q and R and the messages a, b, c, d.



Exercise 2: Snapshot Algorithm According to Chandy/Lamport

1. P sends m.
2. P initiates the snapshot, stores the state 101, and sends a marker to Q.
3. Q receives m, its state is 102.

4. Q receives the marker and stores its state 102. Thus Q has already received all markers, the recorded state of the message channel from P to Q is empty.

5. Q sends a marker to P

6. Q sends m back to P

7. P receives the marker

8. P stores the state of the message channel from Q to P. This is empty because P did not receive a message before the marker.

Determined state: $P : 101, Q : 102$, both channels are empty.

Note: Q could also execute step 6 immediately after step 3. In this case the state would be: $P : 101, Q : 102$, channel $Q \mapsto P$ contains message m.

Exercise 3: Programming: Snapshot Algorithm According to Chandy/Lamport

You will find the solution to this problem in in the archive `l08eFiles.zip`[1] on the lecture's web page.

Exercise 4: Bully Algorithm

a) All higher numbered processes get two ELECTION messages. They also answer both messages with OK, so that both initiators cancel the election. If a process receives an ELECTION message but already holds an election, it ignores this (redundant) message. The higher numbered processes ignore the second ELECTION message and perform their election as usual.

b) First, note that this is unwanted behavior if there is no direct advantage in using a higher numbered process, because a re-election is usually wasteful. However, the numbering of processes can reflect their relative utility (for example, higher numbered processes running on faster machines). In this case, the benefit of re-election can be worth the cost of it. The cost of re-election also includes repeated communication, i.e. the rounds in which ELECTION messages are sent, and the transmission of status messages from coordinator to coordinator.

To avoid re-election, a recovering process could only send a requestStatus message to the next successive lower numbered processes. This would allow them to find out if another process has already been selected and only elect themselves if they receive a negative response. The algorithm can then work as before: If the process just recovered discovers that the coordinator is failing or is receiving an ELECTION message, it sends a coordinator message to the remaining processes.

c) With the prerequisites mentioned in the question, we cannot guarantee that a unique process can be chosen at any time.

Instead, we can accept it as sufficient to form subgroups of processes that agree with their coordinator as such and allow several such subgroups to exist at once.

For example, if a network is divided into two, we could form two largest possible subgroups, each of which would choose the process with the highest identifier among its members as the coordinator. However, in the event that the partitions are merged again, it should be ensured that the two groups become one with only one coordinator.

Hector Garcua-Molina's so-called invitation algorithm achieves this. He chooses a single coordinator within a subgroup whose members can communicate with each other. At regular intervals, the coordinator requests other members of the entire set of processes to try to merge with other groups. If another coordinator is found, a coordinator sends an invitation message to invite him to form a merged group. If a process suspects its coordinator's inaccessibility or failure, it performs an election, as in the bully algorithm.

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/vs/l08eFiles.zip`

Exercise 5: Ring Algorithm

When a process receives an ELECTION message, it checks who sent it. If it has sent it itself (i.e. its number is at the beginning of the list), it converts the message into a COORDINATOR message, as described in the lecture. If it has not sent an ELECTION message, it enters its process number and passes the message on in the ring.

However, if it has previously sent his own ELECTION message and has just discovered a competitor, it compares the process number of the originating process with its own. If the other process has a lower number, it discards the message instead of passing it on.

If the other process has a higher number, the message is forwarded as usual. In this way, only the ELECTION message with the higher number remains when multiple ELECTION messages are sent. The rest is discarded along the way.