IN4150 Lab Exercise 3c report

David Hoepelman: 1521969

Algorithm description

For this exercise Afek and Gafni's election algorithm for asynchronous complete networks was implemented.

The algorithm provides a way to elect a node inside a network. Every has an **ordinary process**. An OP can be captured by a **candidate process** (CP).

Whenever a Node wants to start an election it spawns a CP which then tries to capture the OP's one-by-one in arbitrary order. A Node is elected if its CP has captured all OP's.

The efficiency of the algorithm comes from the capture mechanic. Every CP has a **level**, which is the number of nodes it already captured. If a CP tries to capture an uncaptured OP it success and the CP becomes the OP's **owner**. If a OP is already captured a comparison is done between the owner's level and the new potential owner. If the potential owner has a lower level it is ignored, which effectively halt that CPs capturing process and prevents it from being elected and generating more messages. If the potential owner has a higher level than the existing owner the OP will try to **kill** its previous owner, stopping its capturing process and preventing it from being elected and generating more messages. If the previous owner is successfully killed the potential owner becomes the new owner.

If the level is tied, an arbitrary unique process id is used as a tie breaker.

Implementation details

The node is implemented in the *Node* class, which has inner classes *Process* for the OP and *Candidate_Process* for the CP. Node has an RMI interface which is used for all communication. The messages are of the *Message* class and contain the sending link and a *(level,id)* pair used in the algorithm. A message can be sent to either the CP or OP of a Node because there is no inherent way to distinguish the message type (e.g. a capture and ack can have identical contents).

When a message is received it is delegated to the CP or OP *process* method. The code of these methods closely resembles the lecture notes pseudocode.

Testcases

I could think of 4 different testcases, of which some were executed with different parameters.

- **Simple**: 1 node starts an election
  Expected behavior: CP of node captures all nodes 1 by 1
- **Concurrent**: Multiple nodes start the election concurrently in the same order
  Expected behavior: CP's with lower process id's almost immediately halt because the first OP is already captured, or are almost immediately killed because the CP with the highest id captures the first node.
- **Slow:** A node starts the election. Another node start an election a little while later
  Expected behavior: the second CP halts when it tries to capture an OP already owned by the first one. If it has already captured nodes it is killed when the first CP captures it.
- **Clash:** *N* multiple nodes start the election concurrently in a different order
  Expected behavior: all nodes capture approximately (1/N) part of the nodes. All but 1 CP's eventually halt because they try to capture an OP which they cannot capture or because they are killed.

The following table contains the results of the 6 testcases I executed, with their type, parameters and results.

Testcase type	N	Parameters	# capture msg	#kill msg	# ack msg	levels reached	Observed behavior
Simple	3	1 captures all nodes	3	0	3	1=3	1 sucessfully captures every other node sequentially
Simple	100	1 captures all nodes in random order	100	0	100	1=100	1 sucessfully captures every other node in random order
Concurrent	3	1 starts election immidiatly followed by 2.	4	1	5	1=1 or 2 2=3	1 captures 1, 2 captures 1 which kills 1. 2 captures 2 and 3.
Concurrent	3	2 starts election immidiatly followed by 1.	4	0	3	1=0 2=3	2 captures 1. 1 tries to capture 1 but is ignored. 2 captures 2 and 3
Concurrent	10	3 nodes randomly start the election at the same time and randomly capture nodes	10-28	0-9	10-27	winner=10 others=0-5	Behavior varies wildly depending on timing and order
Slow	10	5 starts capturing, and a while later 6 starts capturing	10-15	1-2	10-15	5=10 6=0-5	6 eventually is ignored or captures a node which tries to kill 5, but the kill is ignored by 5, which causes 6 to stop trying to capture more nodes. If 6 already captured a node 5 capture it which kills 6
Clash	3	1 and 2 concurrently start election, and try to capture the nodes in reverse order.	4 or 5	1 or 2	5 or 6	1=1 or 3 2=1 or 3	2 captures 3, 1 captures 1. Then there are 2 cases: if 1 first captures 2: 2 tries to capture 2 and a kill message is sent to 1, but it is ignored (because level is now higher). 1 captures 3 and 2 is killed if 2 first captures 2: 1 tries to capture 2 and is ignored. 2 captures 1 and 1 is killed