Exercises

Homework for the Monday 27th Sept 2021:

1) Exercice about The two variants of the Chang& Roberts algorithm, refer to exercise 1 distributed in class (see otherwise "List of annexed exercises")
The first variant proposes two guarded atomic actions, that are running on each process having identifier "i" (proclamation phase is not addressed here); with i not equal to 0

local variable: M, initially = to 0
Ii: {M==0} M=i; send <M> to the next in the ring
Mi: {a message holding an integer value <j> is ready}
  receive message;
  if (M<j) then M=j; send <j> to the next in the ring .
  if (M==j) then "I am the leader"

Remember that there is a non determinstic choice when guards are true.
Q1) could it be the case that Ii is never run on a given process with id "i" ?
Q2) are all processes mandatoritly candidate to be elected as leader ?

The second variant uses an alternate way to write message-driven algorithms.
Each process with id "i" holds a local variable "participant" that is a boolean, initially equal to false.
In case the process i is awaken "spontaneously", it has the chance to run the code
participant=true; send <i> to the next in the ring
Then, it will wait for any message arrival, and will execute the code below, and is not anymore allowed to run the code above. If it receives a message and has not yet awoken, the process is able to directly run the piece of code below:

When a message holding <j> is ready to be processed :
  if (j>i) then participant=true; send <j> to the next
  if (j<i) and (participant==false) then participant=true; send <i> to the next
  if (j==i) then participant=false; "I'm the leader"

Q3) Is it possible that a process that did not want to be spontaneously a candidate to the election, can turn out to be candidate?
Q4) Write this version using the guarded atomic action format, be careful that it behaves the same as the version proposed
Q5) Conclude: what is the main difference of behaviour of these two variants
In each version, do we always elect the process that holds the highest id in the network ? Do you think it is problematic or not ?

2) Exercice about Election of a leader in an anonymous network: I have proposed a solution at the end of this wiki (see exercice number 5), and, I simply ask you to check if it works (for me, it works, but, not sure, maybe one of you find a bug ?). Evaluate the expected number of steps/rounds, and, the total expected number of messages generated by the proposed solution. In case you propose an alternate algorithm because you find the one proposed is not suitable, please, evaluate the complexity of your proposed version.

Former Exercises given as Homeworks or for self-training

1)**Exercise inspired by the Zookeeper "virtual shared memory" algorithmic model**

Start documenting yourself about the virtual shared memory model defined by Zookeeper (abstraction of globally shared, hierarchically organized znodes). http://zookeeper.apache.org/doc/current/zookeeperOver.html

Then, consult at the following URL, how **Election in Zookeeper** could be designed.

http://zookeeper.apache.org/doc/current/recipes.html#sc_leaderElection

Add the **Proclamation phase** that is mentioned as missing.

Compare this approach with the exercise about a client(s) server based "topology/architecture" to elect a leader.

2) **A Simple, unidirectional ring-based algorithm to count the total number of processes**

Assume there is a ring of N connected processes (as in Chang&Roberts). Assume there is one of the process that is currently the leader. The leader, at some point, wants to count how many processes are part of the ring (including itself). For this, devise an algorithm, inspired by the one of Chang&Roberts, that will allow the leader to get the total number of processes, and then, to "proclaim" the result to all other processes in the ring.

You must avoid that non leader processes initiate such a counting process, as , we want that the counting is started only by the leader. This means that you must be careful that processes that need to be executing to allow the counting to be executed, are not allowed to trigger themselves a counting procedure. They only must be ready to participate in the counting algorithm, and, eventually, receive in a specific tagged message, that proclaim the result of the counting (as , they need at some point in the application layer, to know how many they are on the ring).

You must then evaluate the complexity of the proposed solution. Is there any best case or worst case ? Or, just one single case to consider ? Give the total number of messages that need to be exchanged among processes (including for the proclamation of the result phase), and also, evaluate the parallel time complexity.

3) **A relevant use of the Probe Echo algorithm**

This exercise is an application of the Probe/Echo algorithm for any topology. See this link to get an example of the general structure of the Probe-Echo algorithm. The applications is from an initiator, (say A) to get a **complete view of the network topology and associated bi-directional links performances**. Assume the topology of the network involves some nodes (6 in the example below), each link is bi-directional and the measured in-coming bandwidth performance is indicated. Each node locally (and initially) knows the name of its neighbours, and the measured performance of the link that connects this neighbour to it. For instance, node A knows it has neighbour B, and the measured bandwidth from B to A as measured on A is 1. It also has node C as neighbour, and measured performance is 2. Here, on each node, is the local information

(source node id, destination node id, performance of the link as measured on the source node) on Node A: (A,B, 1) (A,C, 2) on Node B: (B,A,2) (B,C,2) (B,D,4) on Node C: (C,A,3) (C,B,3) (C,D,3) on Node D: (D,B,1) (D,C,1) (D,E,3) on Node E: (E,D,1) on Node F: (F,D,2)

The goal is, say on node A the initiator, to collect all the triples enumerated above. I.e. on this example, node A could receive two echo messages, one from B, one from C (or alternatively but perhaps less probable, one echo from B and a probe message from C). Depending on the way the probe messages visited the sub-graph rooted at respectively B and C, it could be the case that the first echo message received from B includes only information related to B, whereas the echo message received from C includes the aggregated information coming from the whole spanning tree rooted at C ie, from D, E, and F. In any case, the resulting output from A when it terminates has to be an aggregated set of such triples without any duplicate.

Your algorithm must be written using the notation seen in the course, with guards, & block of actions. A local boolean variable initiator is declared on each process, and it is only true on one process. This means a same piece of code is deployed on each process, but in order to make sure that it executes on the initiator only, you write it like this:

Ci: {initiator=true and xxx} code to run

You can tag your messages, like e.g. "probe" or "echo", and if you want a piece of code only executed for a echo message, you can have as guard

REi: {a message of type "echo" is ready to be consumed on Process i and contains as information a field named "set_triples"} receive the message **;** rest of your code, that extracts "set_triples" from message payload and does what is needs to do with it

Provide both **your complete algorithm**, and **one possible execution** on the network provided as example. Show explicitly which probes and echos messages are exchanged. Is it the case that each bi-directional link is traversed by exactly one probe and one echo message, each in the opposite direction ? **Whether it be yes or no, argument your reply**.

YOU FIND HERE THE CORRECTION provided by one student !

4) **Tel election algorithm**

Sketch a possible execution of the Tel Election leader algorithm, on the following graph of nodes, assuming communication links are bi-directional. You can read on the algorithm code, and from the exercise description, that we will elect the leader whose identity is the largest. The algorithm for this election for any kind of topology, is described from page 3 of this document, section 3.2.

First write it down on the paper that you will give back to me, so to make it clearer than it is. In particular, take the chance to correct the typos that are indicated as comments in the PDF: these typos are about the leader_trouvé message type, that at some places should replace the chercher_leader message type. Explain why we need this additional message type, what role(s) messages of this type play in the whole algorithm. For this, refer to course at slide number 16. Explain why we could remove the parameter K in the messages tagged "leader_trouvé".

Secondly explain why we don't need an explicit echo message type as we had in the Probe Echo algorithm ? However, this echo is somehow still needed, but, how do a process knows that it has to send an echo to its father ?

To make things not too long (for you, and for me), first, exemplify how a Echo algorithm execution initiated by node F execute. Then, do the same from node B. Then generalize by synthetically describing, assuming all nodes spontaneously start the election process, how this algorithm elects F as the leader. As we have done in the course, none of the process has a global knowledge of the whole network (i.e., it just knows who are its neighbours it can send or receive messages from).

Here is the graph description:

(source node id, destination node id) => on Node A: (A,B) (A,C); on Node B: (B,A) (B,C) (B,D); on Node C: (C,A) (C,B) (C,D); on Node D: (D,B) (D,C) (D,E) (D,F); on Node E: (E,D); on Node F: (F,D);

You will find a correction provided by one of the 2018 cohort of students HERE . However, nobody perfectly replied to the B' simulation of the TEL election. Indeed, the guarded action for receiveing a message includes the boolean engaged (which must be true). This means that if a process that did not spontaneously proposed itself as candidate already receives a candidate message, eg. C with B as candidate, it will **first** execute the Initialisation action and propose itself as candidate before processing the received message(s). This means B' tentative will end up once B' message is processed on C (but many of you replied that at the end , no leader gets elected which is wrong). But this also means that all, including F, will propose as candidate, so, at the end F will also be elected!

5) **A Simple (central) server based election algorithm** (still, distributed but not fully peer-to-peer algorithm)

Define a client(s)-server message-passing based approach to conduct an election. The server process that hosts "the election service", is the receiver of the information sent by candidates which are client processes. Describe the respective server and client message-based algorithms. You will have to devise an algorithm where the elected process is **not** mandatorily (as usual) the one that has the greatest identity in the network. Indeed, remember that we do not really care who is elected ! The only thing we care is that one and only one is elected and that all agree about the election outcome once it has been elected.

Do not forget to define how the server proclaims the result of the election, i.e. how a client is informed of the result of the election, be it candidate or simply willing to know who is currently the leader. As we have done in the course, none of the processes has a global knowledge of the whole network (i.e., the server process **does not even know how many client processes** could contact it to be elected). The topology might be considered here as a star, where the server stands in the center, and each client is able to communicate with this server process; but the server does not know how many clients know its identity: this is why we assume the server does not know how many clients there are in total.

We can assume there is an external knowledge (an oracle), that detects if the current leader has failed (and put the current leader value stored at server side and at each client side as empty), and that a new election must take place. Sketch a simple manner for any process or for the oracle, to detect that the current leader identity is not anymore valid.

Write the algorithm (one for the server, another for client processes) using the guarded actions syntax. In the algorithm, do not force a client to be a candidate, and get elected if it is not its intention. However, we assume that there is at least one process that is willing to take the leader role and that is going to be eventually elected. Include in your algorithm what is needed to proclaim the result of the election. And make sure your algorithm is able to correctly treat any client request -as candidate or not-, even if the election is already done and in the process of being proclamed.

What is the cost (in terms of parallel time, and total number of exchanged messages) of this algorithm. For this complexity evaluation, assume that all client processes (still alive in the topology) are all candidates to be elected.

Discuss about drawbacks and advantages of this clients-server approach.

6) **Relevant use of the Probe/Echo Algorithm** :

Design a specific application of the Echo algorithm presented in exercise 3 below, to **count (by suming) total number of processes** in a network whose topology is an arbitrarily connected graph. Take the Echo algorithm sketched here and further commented (in English) on pages 1 and 2 of this document.

Each process is supposed to have an identifier, but you can (must) solve the problem without taking advantage of this knowledge. The only assumption about the network that is needed is which are your neighbour processes. Be careful in this algorithm, because the graph is not mandatory a tree... meaning that you can receive, as in the Echo algorithm, an "info" message but more than once, as you may have more than one neighbour. So be careful in this case! The algorithm starts when an initiator node (assume we have exactly one such) sends out an "info message" to each of its neighbors, and ends when the initiator has received an echo or an info from each neighbor holding a value that it will sum. When the algorithm terminates, the initiator of the algorithm should figure out the total count. Also, do not forget to define which precise sort of information an "info" or an "echo" message must propagate. Briefly argue why your solution gives the correct total number.

Hints to solve the List of Exercises distributed/printed with the course

Some info about exercice 2: provide a possible execution of the Routing Table computation given the provided graph, and taking ideas from algorithm page 8 of Mattern article. You will find here the correction for this exercise 2.

TRY alone to do **exercise number 3** about a generalization of Chang and Roberts election algorithm for any connected graph. Take the Echo algorithm sketched here and further commented (in English) on pages 1 and 2 of this document , use it from any process. You can eventually have a look to the correction from page 3 of this document.

Train yourself on the interesting **exercise number 4** (take care of using all assumptions given in the instructions). Read the first part of this partially correct solution document. BUT THIS SOLUTION provided in 2011-2012 was INCORRECT. **So, at the end of the document (of this incorrect solution), you find additional assumptions that can help to get a correct solution**. This constitutes the newest version of the exercise.

exercise number 5 which I recall here: (from Ghosh book, chapter 5) Consider an **anonymous** distributed system consisting of N processes. The topology is a completely connected network and the links are bidirectionnal. Propose an algorithm using which processes can acquire unique identifiers. (Hint: use coin flipping, and organize the computation in rounds).

Some more explanations: You can provide a solution that is terminating after a number of rounds that is eventually high with a high probability. I think the interesting aspect of this exercise as a training effort for this chapter, is coming from the fact that I'm asking to organize the algorithm in rounds. So give precise information regarding how the processes are aware that the current round has finished. You will assume that the topology is a complete graph which means that any process has N-1 in-going and out-going (so, bidirectional) communication channels (so that means that the total number N of alive processes that run the election algorithm is known). Even if real IDs of processes are never shown, a process can identify from whom a message is arriving by knowing the channel ID. This means that if at the end, a process receives from channel number C indicating that process sending this message on C claims that it is the elected leader, this is a way to memorize who is now the leader that has been elected. This means that in the future, if the process needs to send a message to the leader, then it simply will send it using channel number C.

Variant of this exercice to solve for Sept 27th 2021 : I propose below an algorithm, and as an exercise, analyse it, and evaluate its complexity
Assumption: all processes known N, the total number of processes in the network; ie for instance, they can know how many incoming/outgoing communication channels they have, and we assume the topology is a fully connected graph

Round 1: all flip a coin, it gives locally a 0 or a 1. Each process broadcasts the information about the result (it is 0 or 1) on all its outgoing channels. After each process has received such message from the N (still active) processes (participating in the election itself), each process sets in a local variable "Count", the total of processes that have chosen the value 1 at the current round.
Next round: only processes that did chose 1 in the preceding round will continue to try to be elected. The other processes will only drop the incoming broadcasted messages.
Round 2: the processes that did chose 1 in the preceding round continue; they again will flip a coin. And send to all other processes (broadcast) a message telling the output of this random choice, at round 2. Again, each process that is still trying to be elected, will wait to have received a number of messages that is equal to the number of counted processes of the preceding round. Once this set of messages has been fully received, this will give a new value for Count, and then, only processes that did choose the value 1 in this round will be able to continue in round 3.
It could happen that the number of processes that eventually decided the value 1 at this round is equal to Count, i.e. no process will be able to quit the election at this round. We can expect that with high probability, a fraction of Count processes will have chosen 0 , and not all have chosen 1. So, there should be less and less processes being active from round to round.
If on the contrary all Count processes have decided 0, this would mean that all quit the current round. So that would stop the election. To avoid such a situation, in case counting number of processes that have selected 1 in the current round is found to be equal to 0, the algorithm would run the same round again, and none of them would quit the round.
And so on....

At some point, there may be only Count=2 processes remaining, that flip a coin, and eventually, one of them will quit by choosing 0 alas the other has chosen 1. The one that remains is concluding that it is the leader.