Distributed Systems: Practical 2

1 About this practical

This is the second of two practical exercises for the Distributed System course. The two practical exercises are equally weighted and together they account for 25% of the marks for the course. This is an *individual exercise*, not to be undertaken in a group.

The course policy on late submission is that late submissions will not be accepted unless arranged through your Director of Studies.

This practical has two parts. The first involves implementing a simulation of the **Ring-based election algorithm** (20 marks). The second part involves implementing a simulation of the **Bully election algorithm** (30 marks).

2 Description of the practical

The aim of the practical is to complete the Java implementation of a distributed systems algorithms simulator to support ring-based and bully election algorithms.

2.1 Getting Started

Make your own copy of the files in the directory

`/home/cs4/www/disys/coursework/prac2`

During this practical you have to modify the files `RingNode.java` and `BullyNode.java`. The other six files, `Controller.java`, `Simulator.java`, `Network.java`, `Channel.java`, `Node.java` and `Message.java` are for you to use unchanged. Compile the files you copied and then issue the command `java Controller`. A simulator window will appear on the screen, composed of:

- A canvas allowing the user to construct a network of nodes and channels.
  - To create a node, left-click on a point in the canvas.
  - To create a unidirectional channel, right-click on a sender node and then right-click on a receiver node.
  - To specify a node as a starter of a distributed algorithm, left-click it.
  - To put a node in a **failed** state, middle-click it.
- A button labelled **Ring-based Election**. Click this button to run the Ring-Based Election algorithm starting the election from the specified starter node. It is your responsibility to provide the implementation for simulating the Ring-Based Election algorithm as described in Part 1 (below).
- A button labelled **Bully Election**. Click this button to run the Bully Election algorithm starting the election from the specified starter node. It is your responsibility to provide the implementation for simulating the Bully Election algorithm as described in Part 2 (below).

2.2 Simulator API

You are provided with a set of methods in classes `Node` and `Message` that you will find useful in your implementation.

Methods in class `Node`:

- `int getId()`, returns the Id of this `Node` object.
- `Vector neighbours()`, returns all the `Node` objects that this `Node` can send messages to (i.e. there is a unidirectional channel from this `Node` to each of the returned `Node` objects.
- `Vector allNodes()`, returns all the `Node` objects in the network.
- `void receiveMessage(Message msg)`, called whenever the `Node` is sent a message. Currently this method contains an empty statement. You should override this method in your implementation.
- `void discardMessage(Message msg)`, removes a `Message` object from the network.
- `void changeGraphicalState()`, changes the colour of the graphical representation of this `Node` to red.
- `boolean isFailed()`, returns true if this `Node` object is in a **failed** state.

The following methods from class `Message` will also be useful:

- `public Message(String type, int value, Node sender, Node receiver)`, a constructor to create a `Message` object specifying the `type` of the message, its `value`, its `sender` and its `receiver`.
- `String getType()`, returns the type of this `Message`.
- `int getValue()`, returns the value stored in this `Message`.
- `Node getSender()`, returns the sender `Node` object.
- `Node getReceiver()`, returns the receiver `Node` object.
- `void send()`, causes the `receiveMessage()` method of the `Node` object `receiver` to be invoked. Graphically, the message will travel from the sender node to the receiver node.

2.3 Part 1

The aim of this part of the practical is to implement the simulation of the **Ring-Based Election** algorithm described in Handout 12. You have to expand class `RingNode` which `extends` the `Node` class. `RingNode implements` the `Runnable` interface. Thus, objects of class `RingNode` can be instantiated as threads. Specifying a starter node and then clicking the Ring-based Election button on the simulator window starts the simulation run. The `Node` objects in the network are mapped to `RingNode` objects and the `run` method of the starter `RingNode` object is called. You should complete the implementation of the `run` method of the `RingNode` class, simulating the `Ring-Based Election` algorithm and changing the coordinator Node colour to red. Note that you have first to check that the network is a ring.

2.4 Part 2

The aim of this part of the practical is to implement the simulation of the **Bully Election** algorithm described in Handout 12. You have to expand class `BullyNode` which `extends` the `Node` class. `BullyNode implements` the `Runnable` interface. Thus, objects of class `BullyNode` can be instantiated as threads. Specifying a starter node and then clicking the Bully Election button on the simulator window starts the simulation run. The `Node` objects in the network are mapped to `BullyNode` objects and the `run` method of the starter `BullyNode` object is called. You should complete the implementation of the `run` method of the `BullyNode` class, simulating the `Bully Election` algorithm and changing the coordinator Node colour to red. Note that you have first to check that the network is fully connected.

2.5 Deadlines and submission

The deadline for completion of this practical exercise is

Wednesday, 13th March 2002 at 5pm.

Please submit the following files only:

- `RingNode.java` containing the definition of your modified RingNode class.
- `BullyNode.java` containing the definition of your modified BullyNode class.

Please submit the above files via email to Yussuf Abu Shaaban (**yna@dcs.ed.ac.uk**). Your code should be submitted as a gzipped TAR file of your directory.

2.6 Assessment procedure

Your Java submission will be tested by being compiled and executed. Credit will be given to good use to the methods outlined in Section 2.2. A submission which uses proprietary (e.g. Microsoft) Java extensions and fails to compile with a version of the JDK will lose credit.

2.7 Queries and clarification

If you have any uncertainties about the requirements of the practical exercise, please contact Jane Hillston by email (**jeh@dcs.ed.ac.uk**). If you have any technical questions, please raise them with your tutor.

Date of issue: Monday 18th February 2002.