CS4103: Distributed Systems

Assignment: P2 - Practical 2 - Programming practical

Deadline: 22nd April 2019

Weighting: 50% of coursework

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

1 Objective

This practical is aimed at designing and implementing a simple distributed system, making use of coordination, and resource access algorithms.

2 Competencies

- Develop experience in implementation of distributed systems
- Develop and consolidate knowledge of leader election and mutual exclusion in distributed systems
- Develop a simple application that uses the capabilities provided by the system

3 Practical Requirements

This document includes a general initial overview and a set of incremental steps. Please ensure that you complete fully one step before moving to the next step.

3.1 Introduction

We are tasked to develop a distributed system for a social networking service. The system is formed by a core set of server nodes which provide the infrastructure for exchanging messages, and some client nodes representing the users wishing to exchange messages. In this spec, we refer to server nodes as **nodes**, to client nodes as **clients**, to a message between nodes as **message** and to a personal message in the social network between clients as **post**. The system has the following components (see Figure 1):

- A group of six server nodes communicating by exchanging messages. Each node receives and routes various posts exchanged in the social network.
- An elected coordinator/leader among the servers in charge of group membership. The coordinator should have the highest ID among the group of servers. The coordinator also keeps track of the available nodes in a list P and forms and maintains a ring-shaped network among the nodes.
- A resource P (e.g., a file), in the form of a list that stores information about the group members. This is accessible by all servers, but managed by the current coordinator.
- A resource Q (e.g., a file), in the form of a FIFO ordered queue that stores posts exchanged in the network, accessible by all nodes in a mutually exclusive way.
- At any one time, each of the six server nodes could be contacted by at most two clients intending to send/receive personal messages to/from other clients.

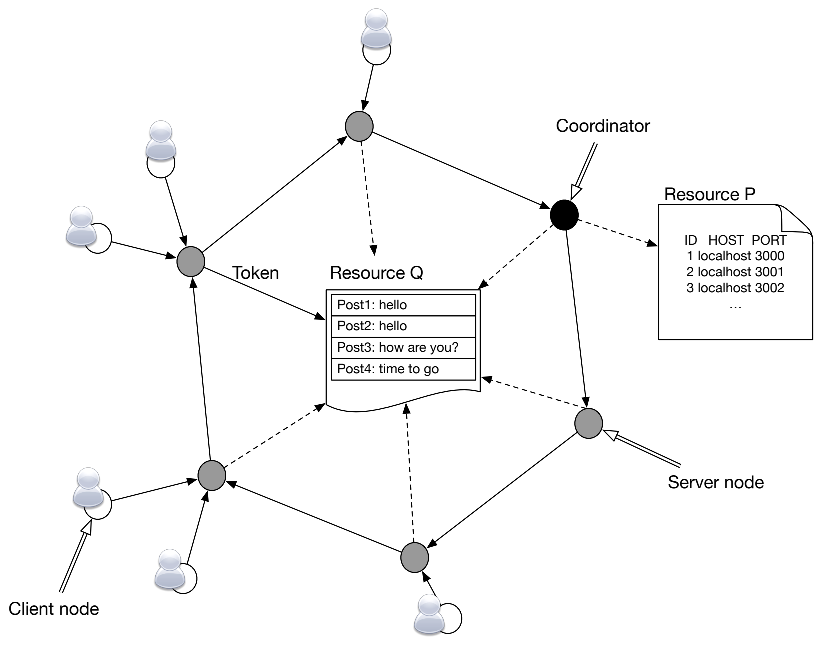

Figure 1: Social network example

To perform these operations the system makes use of the **ring-based algorithm** for leader/coordinator election and of the **token-ring algorithm** for mutual exclusion.

3.2 Part 1: Initial set up

Communication. Create a system with 6 nodes communicating using **available Socket APIs only**. The initial set-up is shown in Figure 2-left. Assume that all nodes function well and all messages are delivered correctly. Each node should run on a separate host and should be assigned an individual communication endpoint (host+port number) and a unique ID. Note that each node should include a client and a server socket in order to contact and be contacted by other nodes (see L7_w3.pdf). For testing purposes you can run all your nodes on *localhost* but they should still be capable of operating over the network; i.e., the solution must not take advantage of any shared memory. The resource P should contain a list of nodes, IDs and their communication endpoints. The initial coordinator is node with ID=6, and each node should initially know which node is the coordinator and its communication endpoint (e.g. as parameter or user input).

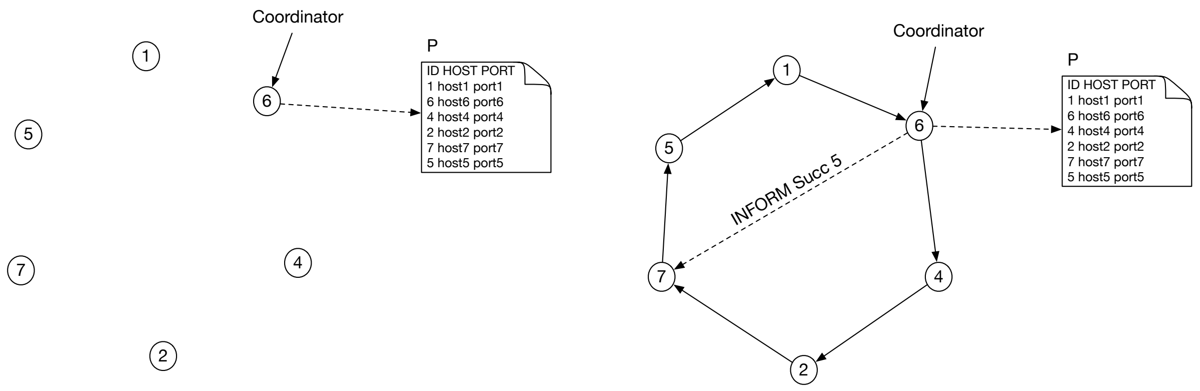

Figure 2: Initial Set Up

Coordinator. You must design a method for the coordinator to check that all nodes listed in P are available. The coordinator has an additional function: if there is no ring, it should construct one by communicating to each node its successor with appropriate communication endpoints. This is shown in Figure 2-right. The ring should NOT be required to be ordered according to the node IDs. You may also choose to form the network as and when a new node is discovered to be alive. After the ring is formed, non-coordinator nodes are only able to communicate with the successor in the ring and the coordinator.

Logs. Each node should keep a log file recording the main activities that it performs including messages that it receives and sends. For Node with ID=5, see some examples of logs below:

```
[2019-03-2 12:03:01:01 | 5 ] Receive from 7: INFORM successor 1
[2019-03-2 12:03:02:01 | 5 ] Ring connection to 1
....
[2019-03-2 12:04:05:01 | 5 ] Receive from 7: ELECTION {7,2}
[2019-03-2 12:05:01:01 | 5 ] Send to 5: ELECTION {7,2,5}
[2019-03-2 12:05:05:01 | 5 ] Receive from 7: COORDINATOR {7}
....
[2019-03-2 12:06:01:01 | 5 ] Receive from Alpha: POST  "I am here"
[2019-03-2 12:06:04:01 | 5 ] Receive from 7: TOKEN
[2019-03-2 12:06:08:01 | 5 ] Add to Q - from Alpha to Beta "I am here"
...
```

3.3 Part 2: Leader/Coordinator election

In Part 1, a coordinator is already present, but we now intend to run another election since node with ID=6 is no longer the one with the highest ID.

For this step, you are required to implement the **ring-based algorithm** for leader/coordinator election. The coordinator in this network should always be node with the highest ID. The election can be initiated by any node that recognises that its ID is higher than the one of the current coordinator ID after the ring has been formed. Each node should have the ability to become a coordinator. Once the election has been performed, each node should keep track of the new coordinator ID and its communication end-point. For an example on how the election works please see the lecture slides (L12_w6.pdf from slide 10). After the election has been held, the new coordinator should again ensure that all nodes in P are available and there is a ring-based network. In your report, include an example of logs to show that your system can perform an election correctly.

Please note that in the current configuration only one node (ID=7) should start the election. Ensure, however, that it is possible to run many elections (E.g. test by setting the initial coordinator to ID=5).

3.4 Part 3: Receive/Send Posts

In this part, we focus on the use of the system of part 1 and 2 to enable clients to exchange posts. Any server node might serve at most 2 client nodes and clients should have IDs or usernames such that they are recognisable destinations for posts. All posts should be exchanged using the queue Q, in a FIFO order. Q permits nodes to deliver messages in an asynchronous, ordered and persistent way. Nodes must gain mutual exclusive access to Q. As per part 2, add information about this process in the logs. Include an example of this functionality in your report.

Mutual Exclusion. Implement **token-ring algorithm** for mutual exclusion. For an example on how this process works please see the lecture slides (L11_w6.pdf from slide 21). The token should be a special type of message with subject TOKEN, which goes around the ring. The coordinator could send the first TOKEN message. A node that receives the TOKEN may only perform two operations: 1) pull out a single post for one of its clients and 2) add a post to be delivered to another node. Once the operations are performed (if any), the node passes the TOKEN on to the next node.

Post Exchange. Building upon the above algorithm, implement the post exchange functionality. For this, let us consider the example depicted in Figure 3. Assume that we have a client Alpha wishing to send a post "Hello" to client Beta. Alpha connects to node ID=4 and Beta connects to node ID=1. Alpha passes the post to node 4 (P1), and 4 waits until it receives the TOKEN (P2). At this point, 4 inserts the new post into Q with destination Beta (P3), and passes the TOKEN on (P4). When node 1 receives the token (P5), it checks for posts that have Beta as destination, and finds a new post “Hello” (P6). 1 removes the post from Q, passes the TOKEN on (P7), and delivers the post to Beta (P8).

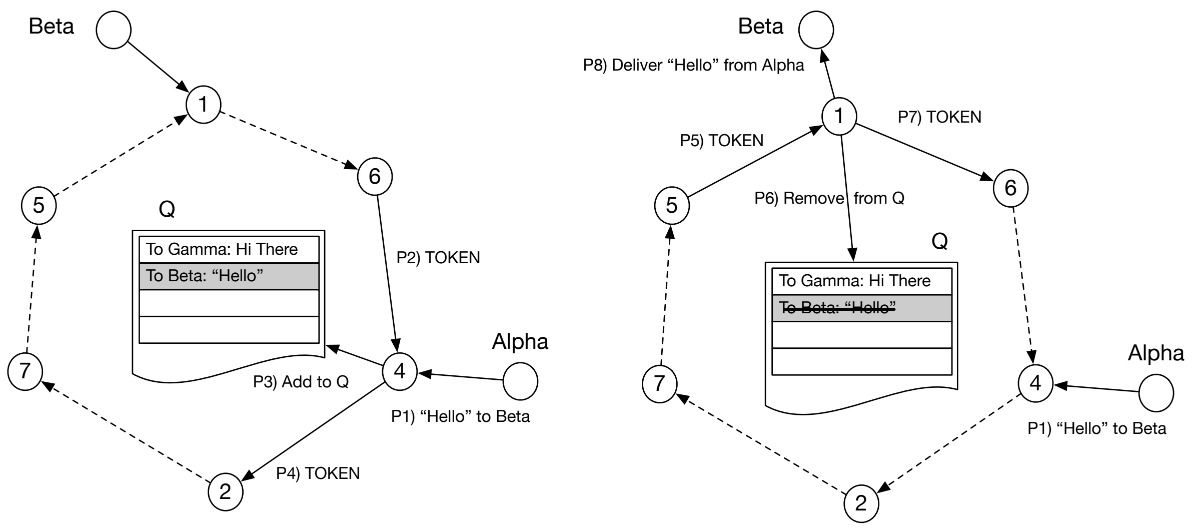

Figure 3: Send/Receive Posts

3.5 Additional Extended Functionalities

It is strongly recommended that you ensure you have completed the Requirements in part 1, 2, and 3 before attempting any of these additional requirements.

You may consider one or two extensions for this assignment. Some suggestions are presented here, but you may present your own extension. Please describe well the purpose of your extensions in the report.

3.5.1 Leaving Nodes

Assume that any node with the exception of the coordinator may fail or leave the network. How do we extend the functionalities of the system to ensure that the system will continue to work, P is up to date, the ring is formed correctly and the token will not get lost in the process?

3.5.2 Joining Nodes

Assume that new nodes can be added in the network if there is a high volume of communication. How do we extend the functionalities of the system to ensure that a new node is ready to participate in the current task?

3.5.3 Multicast messages

In Part 3, we implemented a method to send a message between two clients. Expand this functionality with the ability for a client to multicast the message to every client connected in the network.

3.5.4 Fault tolerance

In Part 2, we implemented a simple trigger for an election in which a node recognises that the coordinator is not the one with highest ID. Expand this functionality with a fault-tolerant mechanism that allows for failures of the coordinator: this should automatically detect the failure of the coordinator, and initiate an election. Note that all other functionalities, including the ring, the P list and the token should be functioning correctly after the election has taken place.

4 Code and Report Requirements

4.1 Code and Libraries

Your implementation must compile and run on the School lab Machines and it should be possible to run your program from the console/terminal without importing it into an IDE. Please develop your program choosing one among the following languages: Java, C or C++, or Python. You are required to develop the algorithms from scratch building upon the available Sockets APIs. You are free to use other libraries as long as they are used for functionalities that are not core to this assignment.

Please include clear instructions on how to compile (if required) and run your system. The solution submitted should contain all source code that you have developed, and should be self-contained. You may use automatic build scripts – eg. Maven. Additionally, you may submit a single solution for parts 1, 2 and 3, depending on what is attempted. If you attempt any of the additional requirements, please separate or clearly mark the extended solution from the main solution.

4.2 Report

You are required to submit a report describing your submission. The report should include:

- A brief list of the functionalities implemented and any extension attempted
- If any of the functionalities is only partially working, ensure that this is discussed in your report
- Design: A description and justification for the functionalities implemented as part of your solution
- Examples and Testing: Selected examples of the main functionalities (screenshots are encouraged) and your approach to testing
- Evaluation: A critical evaluation of the functionalities of your system and what can be improved
- Running: Include clear instructions on how to compile (if required) and run your system (this could also be on a readme file)

The suggested length for the report is between 1000–2000 words.

5 Deliverables

You should submit a ZIP file via MMS by the deadline containing:

- A PDF report as discussed in Section 4.2.
- Your implementation of the solution including any source code (see Section 4.1).
- Any additional script required to run your system

6 Assessment Criteria

Marking will follow the guidelines given in the school student handbook (see link in the next section). The following issues will be considered:

- Adherence to the requirements
- Quality of the solution provided
- Example runs
- Insights demonstrated in the report
- Extension activities completed, if any

Some guideline descriptors for this assignment are given below:

- **up to the band 8-10:** Part 1 fully completed adequately documented or reported.
- **up to the band 11-13:** Part 1 fully completed & some attempt to part 2. The code submitted is of an acceptable standard, and the report describes clearly what was done, with good style.
- **up to the band 14-16:** Part 1 & Part 2 fully completed, with some attempt to part 3. The code submitted is clear and well-structured. The submission includes a clear report showing a good level of understanding.
- **up to 17:** Part 1, Part 2 & Part 3 fully completed. The submission includes code written in good style, together with a clear and well-written report showing real insight into the subject matter.
- **above 17:** Part 1, Part 2 & Part 3 fully completed, and one or more extensions completed. The submission demonstrates unusual clarity of design and implementation, together with an outstandingly well-written report showing evidence of extensive background reading, a full knowledge of the subject and insight into the problem.

7 Policies and Guidelines

7.1 Marking

See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

7.2 Lateness Penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

7.3 Good Academic Practice

The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Alice Toniolo
(a.toniolo@st-andrews.ac.uk)
March 25, 2019