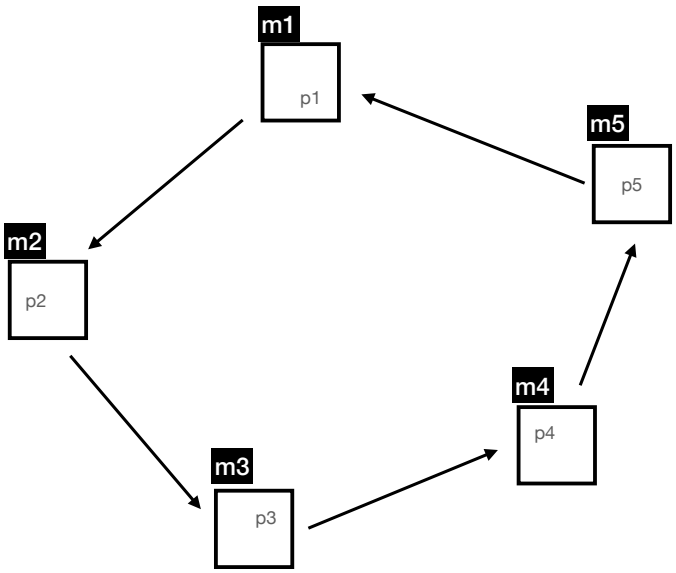
Practical Assignment
Distributed Systems
– 2022/23 –

Starting with the simple templates that have been presented in the practical classes, implement application prototypes for the following scenarios.

1. Token Ring Algorithm



1. you should first read van Steen & Tanenbaum (chap. 6, "Token Ring Algorithm");
2. create a ring network with 5 peers (call them `p1` to `p5`) such as the one represented in the figure above;
3. note that each peer must be in a different machine (`m1` to `m5`);
4. each peer only knows the IP address of the machine where the next peer is located; `p1` knows the IP of `m2`; `p2` knows the IP of `m3`; ...; `p5` has the IP of `m1`, thus closing the ring;

5. the client in each peer has a small shell that executes only two commands: lock and unlock;

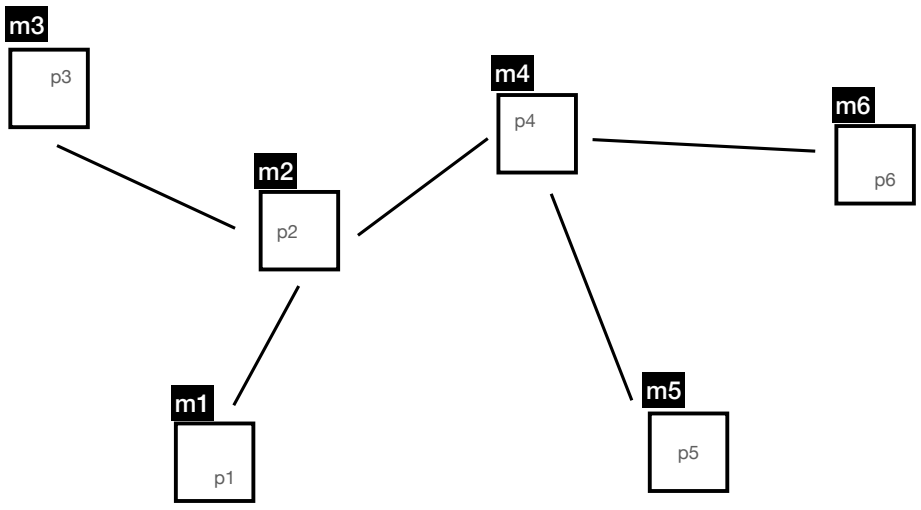
6. another thread in that peer runs in a loop waiting for a message (that can only come from the previous peer); when it receives one, it forwards it to the next peer in the ring;

7. the message (called token in the algorithm) only contains an integer value; this value is initially 0 and is incremented by 1 at each peer that receives it;

8. whenever a peer receives the token, it prints its IP and the token value on the screen;

9. when the command lock is executed in the shell of one of the peers, the token is not forwarded by that peer until an unlock command is executed in that same shell.

2. Data Dissemination via Gossiping



1. you should first read van Steen & Tanenbaum (chap.6, "Information Dissemination Models");

2. create a network of 6 peers (p1 to p6), running on different machines (m1 to m6), with the topology shown in the figure above;

3. the network must be built peer by peer using the register command from different peers (see below);

4. each peer keeps a table with the IPs of the machines/peers that have registered themselves with it;

5. the command - register ip2 - registers the current peer, say p1, with the peer p2 at ip2 (if successful, the IP table for p1 gets ip2, the IP table for p2 gets ip1);

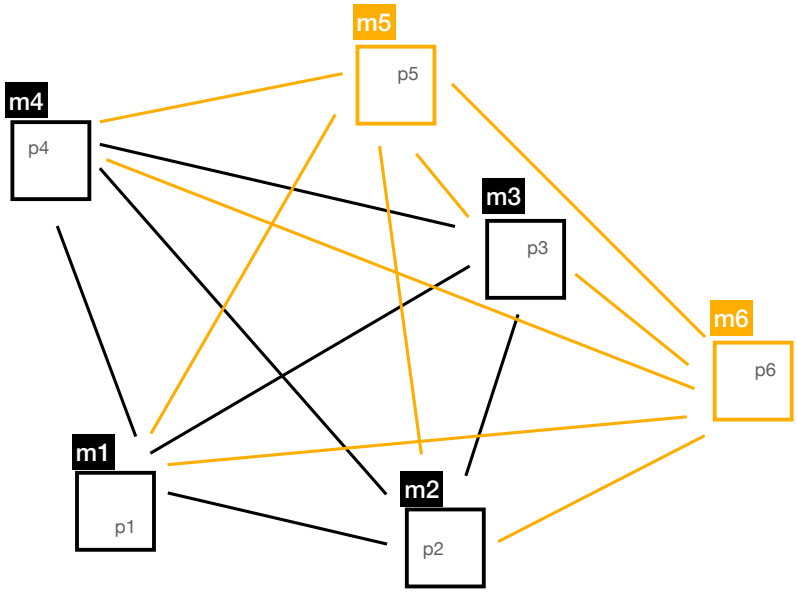
6. each peer also keeps a growing list of words (initially empty); the words are generated randomly as a Poisson process (see the code template provided in Moodle) with a frequency of 1 event each 30 seconds; the new word is added to the list;

7. the words should be selected at random from a file (the same file for all peers - hint: get a text format dictionary from the Web); at any given time, the 6 peers will have a different collection of words;

8. every time a new word is generated by a peer, it should be gossiped to the others; apply the Gossiping Algorithm;

9. note that you should check for repeated messages and stop gossiping with probability $1/k$ (use $k = 5$).

3. Reliable Totally Ordered Multicast



1. you should first read van Steen & Tanenbaum (chap.6, "Lamport Clocks" and "Totally Ordered Multicast");

2. create a network of 6 peers (`p1` to `p6`) each in a different machine (`m1` to `m6`) with the topology shown in the figure above;

3. each peer has a table with the IPs of all the other peers/machines;

4. implement "Lamport Clocks" in each peer so that messages can be adequately timestamped;

5. assume that peers `p5` and `p6` are replicated servers that receive requests from the other peers; e.g., imagine that these two peers are 2 replicas of the same database;

6. the goal is to make sure that both p5 and p6 see the same sequence of messages from the other peers, in other words, that the peers agree on a global order for the messages before they are delivered to the servers;

7. peers p5 and p6 can be implemented as "mock" databases; they just receive messages and print them; no need to implement an actual database;

8. the other peers generate requests to p5 and p6 with a Poisson distribution with a frequency of 1 per second; each request is sent to all other peers (all peers in the IP table);

9. to make sure that all requests issued from peers p1 to p4 are processed in the same order in both p5 and p6 you must implement the Totally Ordered Multicast Algorithm using Lamport clocks to timestamp the messages (check here for another, detailed, description).

General Remarks

This practical assignment can be done individually or in groups of at most 2 students. **The composition of the groups must be sent by e-mail to me until at most October 28th**. Please send the complete names of the group members and the respective numeric identifiers in Sigarra.

I suggest that you use Java to implement the 3 scenarios. You may use Java Sockets or gRPC (or both in different scenarios). If you are keen on using another programming language please check with me first. Assuming you will be using Java, the software you will produce should be organized in 3 packages as follows:

- ds.assignment.tokenring
- ds.assignment.gossiping
- ds.assignment.multicast

To submit your code, please send me an e-mail with subject DS2223:SUBMIT with a .ZIP attached. This file should contain the source code for the 3 packages and a text file README with the name and numerical identifiers of the group members, together with a short text explaining how to compile and to run the examples.

The deadline for code submission is January 6th, 2023. In the week after, each group will make a demonstration of the applications to me (day/hour to be scheduled later). **This demonstration is mandatory and all members of the group must be present**. **Failure to meet this requirement will result in a grade of "0" in the practical assignment**.

Enjoy your work,

Luís Lopes
lblopes@dcc.fc.up.pt