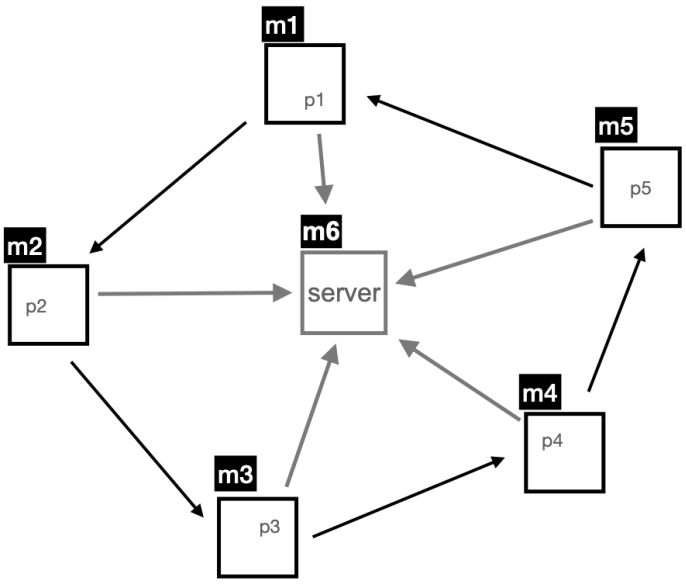
Distributed Systems
– 2023/24 –
Practical Assignment

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

1. Mutual Exclusion with the Token Ring Algorithm



1. read van Steen & Tanenbaum (Chapter 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; each peer must be in a different machine (`m1` to `m5`); besides these 5 peers, create also a `calculatormulti` server called `server` that runs on machine `m6`;
3. 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; all peers know the IP address of the machine where the `server` is located (`m6`); these can be passed to the peer in the command line, e.g., for peer `p2` you would run the following command (in machine `m2`): `$ peer m3 m6`;

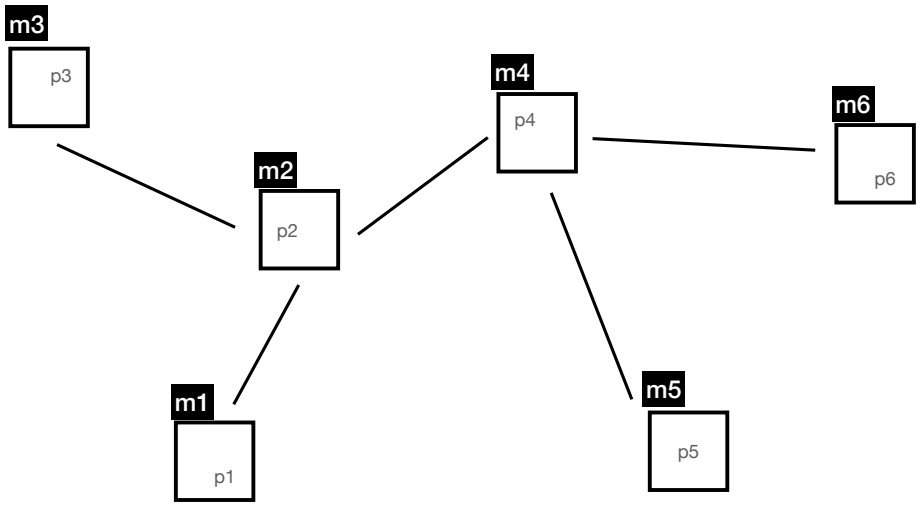
4. one of the threads in each peer generates requests for the `server` following a Poisson distribution with a frequency of 4 events per minute (average); each request is also random (both the operation and the arguments); once the reply from the request arrives the peer must print it in the terminal;

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

6. the token does not have any contents; the peer that holds the token at any given moment has exclusive access to `server` (thus implementing mutual exclusion);

7. thus, whenever a peer receives the token, it checks if it has a request for the server and if so interrupts the forwarding of the token until the request is processed; when the result is received and printed in the terminal, the peer restarts the forwarding of the token and computes the time to the next event.

2. Data Dissemination Using the Anti-Entropy Algorithm



1. read van Steen & Tanenbaum (Chapter 6, "Anti-Entropy Algorithm");

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;

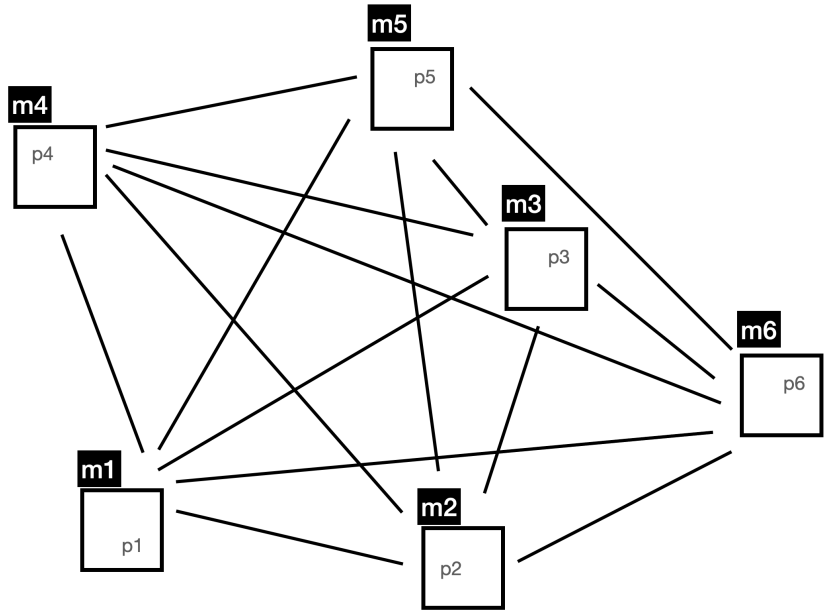
3. each peer keeps a table with the IPs/names of the machines/peers that have registered themselves with it; these IPs/names are provided in the command line when the peer is executed, e.g., for peer `p2` you would run the following command (in machine `m2`): `$ peer m1 m3 m4` whereas for peer `p6` (in machine `m6`) you would run: `$ peer m4`;

4. 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 every 10 seconds; each new word is added to the list;

5. the words should be selected at random (with different seeds) 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;

6. each peer selects a random instant, using a Poisson distribution with a frequency of 1 every minute, to perform a *push-pull* operation with its neighbors.

3. A Basic Chat Application Using Totally Ordered Multicast



1. read van Steen & Tanenbaum (Chapter 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; each peer has a table with the IPs of all the other peers/machines;

3. implement Lamport clocks in each peer so that messages can be adequately timestamped;

4. each peer sends a random message according to a Poisson distribution with a frequency of 1 per second (for the messages, reuse code from the previous problem); each word is sent to all other peers (they are all in the IP table);

5. the goal is that all peers 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 processed; every message received and processed by a peer must be printed in the terminal; the printed list of messages for all peers must be the same!

6. to make sure that all words sent from peers are processed in the same order by all peers you must implement the Totally Ordered Multicast Algorithm using Lamport clocks to timestamp the messages (check here for another, detailed, description).

General Remarks

The practical assignment is individual. I suggest that you use Java to implement the 3 scenarios. You may use Java Sockets or gRPC (or both for different problems). 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 into 3 packages as follows:

- `ds.assign.ring`
- `ds.assign.entropy`
- `ds.assign.chat`

To submit your code, please send me an e-mail with the subject `DS2324` with a `.ZIP` attached. This file should contain the source code for the 3 packages and a text file `README` with a short text explaining how to compile and run the examples. The deadline for code submission is January 5th, 2024. In the week after that, each student will make a demonstration of her/his work (day/hour to be scheduled later). **Please note that the demonstration is mandatory. Failure to meet this requirement will result in a grade of 0 ("zero") in the practical assignment**.

Enjoy your work,

Luís Lopes
`lmlopes@fc.up.pt`