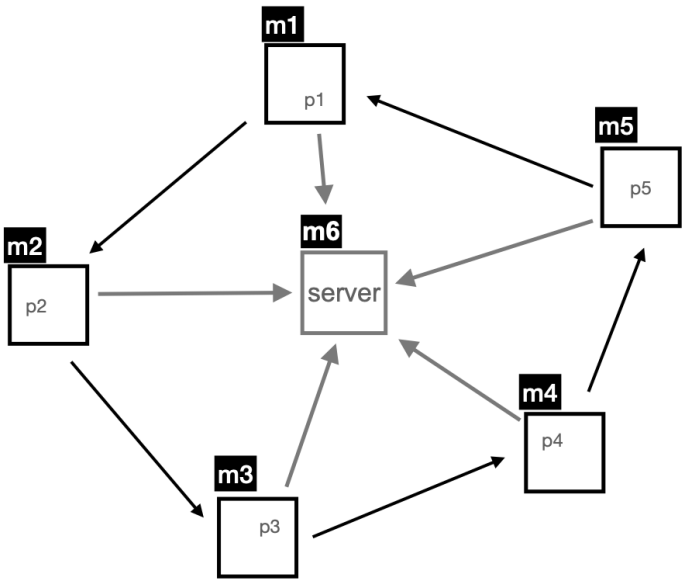
Distributed Systems
– 2024/25 –
Practical Assignment

Start with the simple templates presented in the practical classes and implement distributed applications for the following scenarios.

1. Mutual Exclusion with the Token Ring Algorithm



$\Longrightarrow$ Read van Steen & Tanenbaum (Chapter 6, "Token Ring Algorithm").

$\Longrightarrow$ 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`.

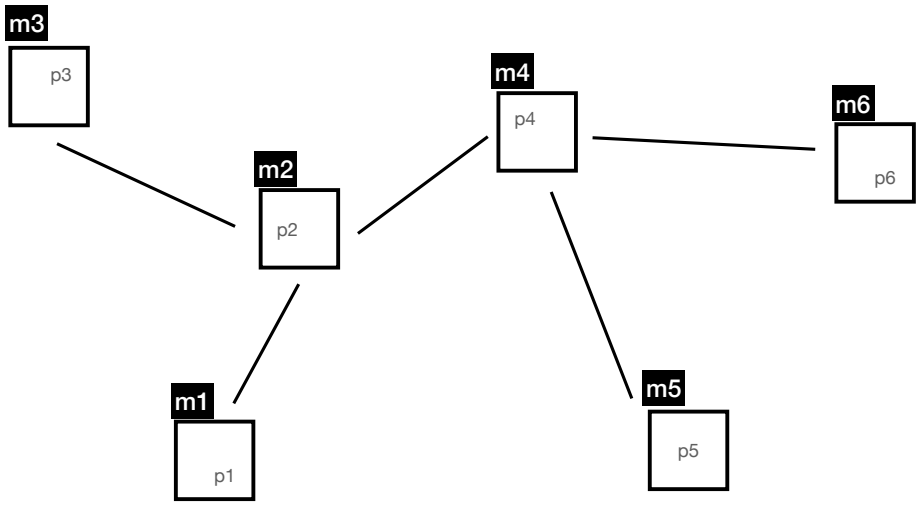
$\Longrightarrow$ Each peer only knows the IP address of the machine where the next peer is located: `p1` knows the IP of `m2`, `p2` knows the IP of `m3`, ..., `p5` has the IP of `m1`, thus closing the ring. All peers know the IP address of the machine where the `server` is located (`m6`). These can 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`.

$\Longrightarrow$ One of the threads in each peer generates requests for the `server` following a Poisson distribution with an average frequency of 4 events per minute. Each request is also random (both the operation and the arguments). These requests are placed in a local queue.

$\Longrightarrow$ 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*. The token can be a void message. The peer that holds it at any given moment has exclusive access to `server`, effectively implementing mutual exclusion.

$\Longrightarrow$ When the peer receives the token, it checks if it has requests for the server in its local queue. If not, it forwards the token to the next peer in the ring. If so, it holds the token until the requests are processed. When the results are received and printed on the terminal, the peer restarts the forwarding of the token.

2. Count the Number of Nodes in a P2P Network



$\Longrightarrow$ Read van Steen & Tanenbaum (Chapter 6, on "Data Dissemination Algorithms").

$\Longrightarrow$ Create a network of 6 peers (`p1` to `p6`), running on different machines (`m1` to `m6`), with the topology shown in the figure above.

$\Longrightarrow$ Peers must have a thread waiting for others to connect and register themselves. This is how peers get to know each other and the network is built.

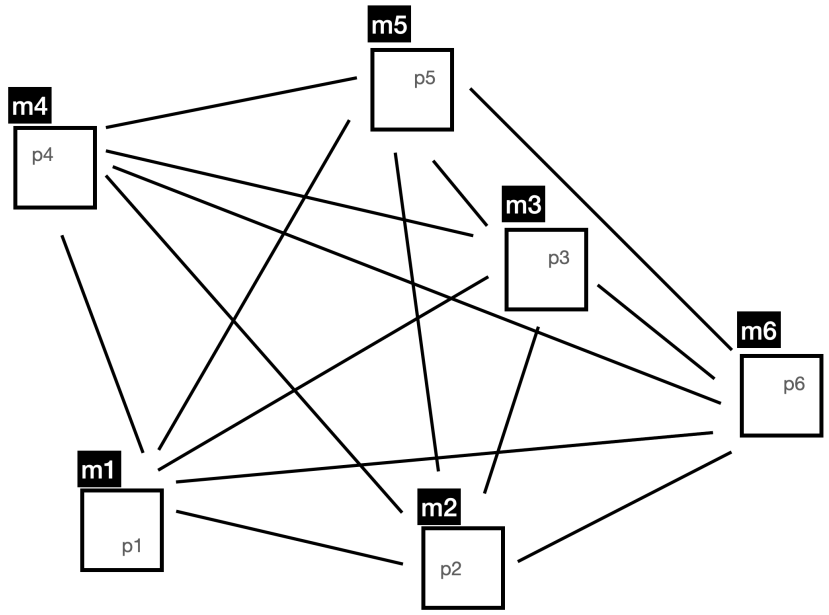
$\Longrightarrow$ Each peer keeps a map data structure (e.g., Map or HashMap in Java) with the IPs/names of the machines of the peers that have registered themselves with it. For example, peer `p2` (in machine `m2`) keeps the entries `[m1, m3, m4]` whereas peer `p6` (in machine `m6`) keeps only `[m4]`.

$\Longrightarrow$ Now implement the Anti-Entropy Algorithm so peers can disseminate their maps. Each peer updates its map twice per minute following a Poisson distribution, each time printing the number of peers in the updated map.

$\Longrightarrow$ Once the dissemination algorithm starts, the number of items in the map at each peer will approximate the total number of peers in the network.

$\Longrightarrow$ What happens to the number of peers if you remove some of them from the above network, e.g., killing the processes? Devise a scheme that handles this situation (hint: you can use timestamps for the map entries and remove entries older than a given threshold).

3. A Basic Chat Application Using Totally-Ordered Multicast



$\Longrightarrow$ Read van Steen & Tanenbaum (Chapter 6, "Lamport Clocks" and "Totally Ordered Multicast").

$\Longrightarrow$ 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.

$\Longrightarrow$ Implement Lamport clocks in each peer so that messages can be adequately timestamped.

$\Longrightarrow$ Each peer sends a random word according to a Poisson distribution with a frequency of 1 per second. You can use a text format dictionary from the Internet and extract its keywords into a set. Then choose a random keyword from that set. Each word is sent in a message to all other peers (they are all in the IP table).

$\Longrightarrow$ The goal is that all peers print the same sequence of words. To do this, the peers must agree on a global order for the messages before they process them (print the words therein). This is not trivial, as factors such as communication latency and varying network topology affect message delivery even in small networks.

$\Longrightarrow$ To achieve this you must implement the Totally-Ordered Multicast (TOM) Algorithm using Lamport clocks to timestamp the messages. Check Chapter 6 of van Steen and Tanenbaum for the details of the algorithm and here for another, detailed, description).

$\Longrightarrow$ Note that in TOM there is a difference between receiving and processing a message. Processes always receive incoming messages but they are processed only when specific conditions are met. In this application, peers process messages by printing the words therein. If you correctly implemented the TOM algorithm the printed list of words must be the same for all peers.

General Remarks

The practical assignment is individual.

I suggest you use Java to implement the 3 scenarios. You may use Java Sockets or gRPC (or both for different problems). If you want to use 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:

$\Longrightarrow$ `ds.assign.trg`

$\Longrightarrow$ `ds.assign.p2p`

$\Longrightarrow$ `ds.assign.tom`

The deadline for code submission is January 3rd, 2025.

To submit your code, please send me an e-mail with subject `DS2425` with a `.ZIP` attached. This file should contain the source code for the 3 packages and a text file `README` explaining how to compile and run the examples. After the submission deadline, you must present your work (day/hour to be scheduled later). To speed things I suggest you develop some scripts that set up the peers and networks automatically.

Not presenting your work results in a grade of 0 ("zero") in the practical assignment.

Enjoy your work,

Luís Lopes
`lmlopes@fc.up.pt`