Synchronization (Vector Clocks) (12 points)

This homework is focused to develop a group messaging application over unstructured P2P overlay network. You can use your Homework 04 as a starting point.

The solution for this homework must be uploaded to OLAT until Monday, **14.05.2018**, at **08:00**, and will be evaluated on 15th of May.

Description of the application.

Users join to the application connecting to a well-known "node", similar as for Homework 04. Assume that after joining (creating) *n* users, the state is convergent and users will never leave. After reaching the convergent state, users send messages to each other in a broadcast manner, i.e. when a user sends a message, all others should see it in the right order. Assume that the network is reliable (messages are never lost) and FIFO. However, the network delays are different for each pair. For example, user *A* can send a message *mA*, user *B* will see it and answers with *mB*. Message *mB* will reach to user C before the message *mA* reaches user *C*.

Using the *vector clocks* technique, develop an application that will keep the conversations synchronized, i.e. each user will see the same conversation. For each user there should be two types of processes: *Application* (*A1*, *A2*, *A3*, …) and corresponding *Middleware* (*M1*, *M2*, *M3*, …). *Application* is a simple process that has two functions: *i*) forwards everything what is written to a console to its corresponding *Middleware* and *ii*) prints to a console each received message from its corresponding *Middleware*. The *Middleware* process is responsible for synchronization by maintaining vector clocks and attaching a timestamp to or reading it from messages. *Middleware* attaches a timestamp to the received message of its corresponding *Application* and broadcasts (floods) it to all other *Middleware* processes, which will deliver the message to its corresponding *Application* only if both conditions are met (it is the "right message" and it is the "right time").

Delays: The different delays should be implemented with a sleep function. Use these delays: *delay(A1, A2)* = 1s, *delay(A1, A3)* = 20s, *delay(A2, A3)* = 3s.

Conversation: *A1* sends message *m1*, *A2* sends message *m2* after receiving *m1* and *A3* sends message *m3* after receiving message *m2*. Be careful, *Ai* is an application process (user consoles), not a middleware's console.

Assume that the clocks at each process are already synchronized.

Assignments:

1. Develop the communication between two processes (*Application* and *Middleware*). **(1 point)**
2. Develop the local vector clocks protocol (send message) at each *Middleware* process. **(1 point)**
3. Develop the part to add a timestamp and reading it at each *Middleware* process. **(2 points)**

Do either 4. or 5.:

4. Develop the message delivery from a *Middleware* to the corresponding *Application* process (deliver the right message at the right time to the corresponding *Application*) according to the use case. **(2 point)**

5. Generalize: Develop a mechanism with a buffer (use some assumption and determine the necessary size per user) that will check if some message in the buffer should be delivered to the corresponding *Application*. **(5 points)**
6. Integrate all developed parts and start three *Application* and three *Middleware* processes. Demonstrate that all three *Application* processes will print all three messages of the conversation in the same (right) order. It is assumed that writing message *m2* will be in a period of time less than 10 seconds. if you need more time, increase *delay(A1, A3).* **(3 points)**

Make assumptions (but some realistic ones ☺ and give a good explanation) if you have difficulties with some part. Use some hard coding if necessary. For example, ports, process IDs, three parameters instead of an array, etc.

For future work:

If you are interested for a possible project and possibly extend to a diploma thesis, please apply. For example, to develop a distributed system that supports arbitrary number of users (Middleware should flood the message to nodes that are not neighbors, as well, with a single copy to each *Application*). Possibly extend with leaving users, fault tolerance and consistency.