Report 3: Loggy - A Logical Time Logger

Michail Roussos, michailr@kth.se

September 27, 2023

1 Introduction

In this exercise we learnt how to use logical time in a practical example. The task, we had to implement, was a logging procedure that receives log events from a set of workers. The events are tagged with the Lamport time stamp of the worker, and the events must be ordered before being written to as output.

2 Main problems and solutions

We had to implement the following modules:

1. The first module we had to implement was the logger module, called *loggy*. In this module we will implement the functionality of the logger. This module will be responsible for the logging process and producing the output.

2. The worker module, called *worker*. In this module we will describe the functionality of the workers that send and receive the messages.

3. A module for the Lamport clocks, called *time*, which we will import into the modules that need to have access to its functions.

4. A module for the Vector Clocks, called *vtime*. This module can be imported instead of the *time* module to have vector clocks instead of lamport clocks.

5. A module for testing the functionality, called *test*.

3 Evaluation

In order to test the functionality of the logger, we can use the test module that was provided.

In this module we create 4 workers that will create messages and a logger that will log them. Each worker will have a clock and upon receiving or sending a message it will communicate to the logger the name of the node, the timestamp of the clock and the message. The logger will keep track of all the messages and upon receiving a message it will add it in a Queue and it will check whether it is safe to print any messages from the Queue. In order to do so the logger must keep track of the timestamps of all the workers and only print a message if it is older than all the timestamps from the workers.

In the Lamport clock case the logger will have a structure like:

```
[{john,3}, {paul,5}, {ringo,2}, {george,7}]
```

In the vector clocks case the logger will have a structure like:

```
[{john,[{john,3}, {paul,5}, {ringo,2}, {george,7}]},
{paul,[{john,1}, {paul,6}, {ringo,3}, {george,2}]},
{ringo,[{john,5}, {paul,2}, {ringo,5}, {george,3}]},
{george,[{john,4}, {paul,6}, {ringo,6}, {george,5}]}]
```

Also in both the *loggy* and the *worker* modules I have added the following order that can be used to change from the Lamport clocks to the vector clocks.

For the Lamport clocks case we want the *time* module

```
-import(time, [zero/1,inc/2,merge/2,leq/2,safe/3,clock/1,update/3]).
```

For the vector clocks case we want the *vtime* module

```
-import(vtime, [zero/1,inc/2,merge/2,leq/2,safe/3,clock/1,update/3]).
```

4 Conclusions

In this assignment we had the opportunity to further our Erlang skills, we saw how a basic logger could be implemented and how to do that in erlang. We also had the opportunity to use Lamport clocks and vector clocks and use them in a real application.