# Loggy: a logical time logger

Zainab Alsaadi

September 29, 2021

## 1 Introduction

The aim of this exercise is to learn how to use logical time in a practical example. The task is to implement a logging procedure that receives log events from a set of workers and print them to stdout in the correct order. The events are tagged with a vector or a Lamport time stamp, which will be implemented and tested in this exercise.

## 2 Main problems and solutions

The main problems were encountered after implementing the modules i.e. in the testing phase. We realize quite early how impossible it is to keep track of events and achieve perfect synchronization.

## 3 Evaluation

First, we have our logger, which task is to accept events and print them on the screen. This is modified so that it prints the events in order.

We have then the Worker module. It illustrates a worker that waits to receive messages and then sends its own messages. The worker has a unique name, access to logger, a seed for the randomized value that is sent in the message, and a sleep and jitter value. The sleep value shows how active the worker is sending messages and the jitter value is a random delay between the sending of a message and sending of a log entry. Both sleep and jitter value are set and evaluated during testing.

A test module is used to set up a set of 4 workers and connect them to each other as peers. Each message is sent to a random peer.
After setting up these given modules, we run some tests, experimented with the jitter and sleep value and found messages that were printed out of order. Knowing that they were out of order is when the ”received” messages with the random value were printed before ”sending”. When setting jitter value

to 70, the sender sends message and then waits before sending a log entry. Meanwhile the reciever sends to the log immediately when receiving a message. Setting the jitter value to 0 eliminated the the wrong-order printout.

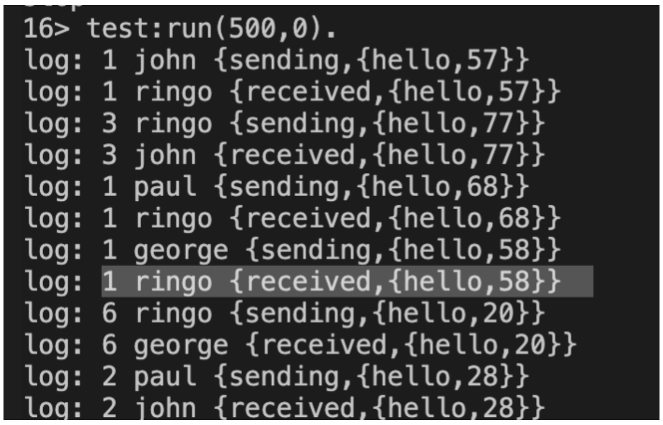
After running this test, the Lamport logical time were introduced to the worker process. The worker can then keep track of its own its own counter and pass it along with the sent messages. To implement the Lamport functionality, the time module was created. This was simply implemented as a counter and running tests showed the result that is seen in Figure 1. It shows the unordered printout of the recieved and sent messages.

The time module was then extended to implement the clock that can keep track of the nodes, the update function of a clock and the checking function if a message was safe to print. The procedure that was followed to know if a message was safe to print was comparing its timestamp to all other processes timestamps. if its less or equal, then it is safe to print. The ones that are not safe to print are kept in a "Hold-back queue" until they were safe to print. Logging now with the timestamp and the ordering procedure showed the results in Figure 2.

After being done with the Lamport time evaluation, a vector clock was implemented as a list of process names and values. Here, each process keeps a vector timestamp, as shown in Figure 3.

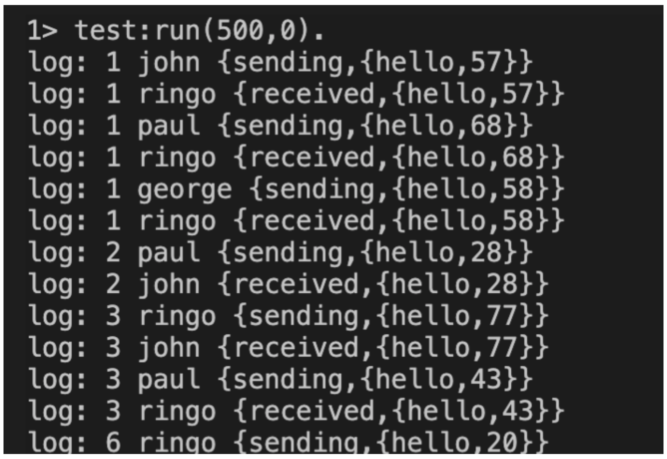
## 4 Conclusions

The presented problem was quite time consuming, but interesting and informative. However, most of the time was spent on implementing the functions according to the description and instructions rather than properly learning the theoretical part.



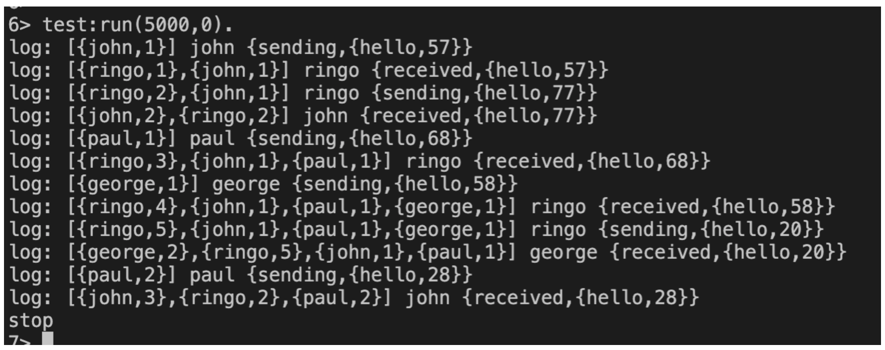
```
16> test:run(500,0).
log: 1 john {sending,{hello,57}}
log: 1 ringo {received,{hello,57}}
log: 3 ringo {sending,{hello,77}}
log: 3 john {received,{hello,77}}
log: 1 paul {sending,{hello,68}}
log: 1 ringo {received,{hello,68}}
log: 1 george {sending,{hello,58}}
log: 1 ringo {received,{hello,58}}
log: 6 ringo {sending,{hello,20}}
log: 6 george {received,{hello,20}}
log: 2 paul {sending,{hello,28}}
log: 2 john {received,{hello,28}}
```

Figure 1: results of running test program before the ordering. Sleep = 500, jitter = 0



```
1> test:run(500,0).
log: 1 john {sending,{hello,57}}
log: 1 ringo {received,{hello,57}}
log: 1 paul {sending,{hello,68}}
log: 1 ringo {received,{hello,68}}
log: 1 george {sending,{hello,58}}
log: 1 ringo {received,{hello,58}}
log: 2 paul {sending,{hello,28}}
log: 2 john {received,{hello,28}}
log: 3 ringo {sending,{hello,77}}
log: 3 john {received,{hello,77}}
log: 3 paul {sending,{hello,43}}
log: 3 ringo {received,{hello,43}}
log: 6 ringo {sending,{hello,20}}
```

Figure 2: results of running test program after the ordering. Sleep = 500, jitter = 0



```
6> test:run(5000,0).
log: [{john,1}] john {sending,{hello,57}}
log: [{ringo,1},{john,1}] ringo {received,{hello,57}}
log: [{ringo,2},{john,1}] ringo {sending,{hello,77}}
log: [{john,2},{ringo,2}] john {received,{hello,77}}
log: [{paul,1}] paul {sending,{hello,68}}
log: [{ringo,3},{john,1},{paul,1}] ringo {received,{hello,68}}
log: [{george,1}] george {sending,{hello,58}}
log: [{ringo,4},{john,1},{paul,1},{george,1}] ringo {received,{hello,58}}
log: [{ringo,5},{john,1},{paul,1},{george,1}] ringo {sending,{hello,20}}
log: [{george,2},{ringo,5},{john,1},{paul,1}] george {received,{hello,20}}
log: [{paul,2}] paul {sending,{hello,28}}
log: [{john,3},{ringo,2},{paul,2}] john {received,{hello,28}}
stop
7>
```

Figure 3: results of exchanging Lamport time module with vector module.