

Homework 3 Report: Loggy

Justas Dautaras

September 20, 2019

1 Introduction

In the third Distributed Systems task a logical time logger is implemented. This application is 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 are then ordered before being shown by the logger.

This task is divided into four parts:

- **Logger** this module accepts events and prints them on screen in the order they happened, using logical time stamps.
- **Worker** these are nodes that are communicating with each other and keeping track of their time. A worker can receive messages and send them.
- **Time** this is a module that deals with managing time issues. This will be covered more later in this document.
- **Test** this module is used to make tests - quickly spawn interacting workers and to keep the logger alive for some time.

Completing this task allows us to test logical time logging properties and understand how it works and where it can be used.

2 Main problems and solutions

One of the main problems given in the first example, where logical time is not yet introduced, is that logging is not sorted. Meaning that because of various delays (in this case fake delays like sleep timers) the logger might receive a message that 'a message X has been received' before actually even receiving that the message has been sent. This is either sorted by removing any possible latency or performance delays. Yet, this is still highly unreliable. This is the problem that logical time solves. In a way. More precisely, what it allows us to do is to put messages in a queue of the logger and only print them when messages time constraints are satisfied:

```

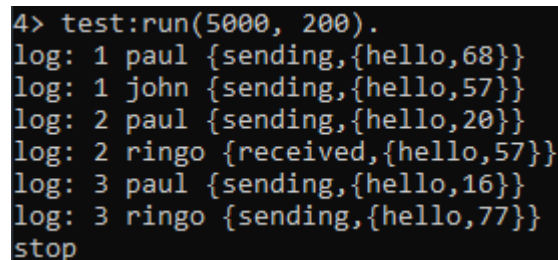
%check whether it is safe to print,
%i.e. minimum time is lower or equal to logger time
safe(Time, Clock) ->
  MinTime = lists:foldl(fun({T,_}, MinTime) -> min(T,MinTime) end, inf, Clock),
  if
    (MinTime >= Time) ->
      safe;
    true ->
      not_safe
  end.

```

Time constraint described above checks whether a message is safe to print - if it's time is below or equal of the minimum time of the clock. All of this is done in 'time' module.

3 Evaluation

Although logical time allows us to be safe before posting logs about whether they have happened one before the other or not. What it does not do is precisely sort the events in the order that they did happen.



```

4> test:run(5000, 200).
log: 1 paul {sending,{hello,68}}
log: 1 john {sending,{hello,57}}
log: 2 paul {sending,{hello,20}}
log: 2 ringo {received,{hello,57}}
log: 3 paul {sending,{hello,16}}
log: 3 ringo {sending,{hello,77}}
stop

```

Figure 1: Output example

4 Conclusions

Solving this task has taught me to program on a more intermediate level of functional programming and helped to understand logical time properties and the fundamentals of distributed systems.