Distributed Systems
Report 3: Loggy

Yerrapragada Akhil
September 25, 2019

Introduction

This seminar is intended to explain the process of creating a logical procedure that receives log events from workers using Erlang and describe the functionality of it. The key technical aspects implemented in the logging procedure are: A worker, logger, Lamport time and test.

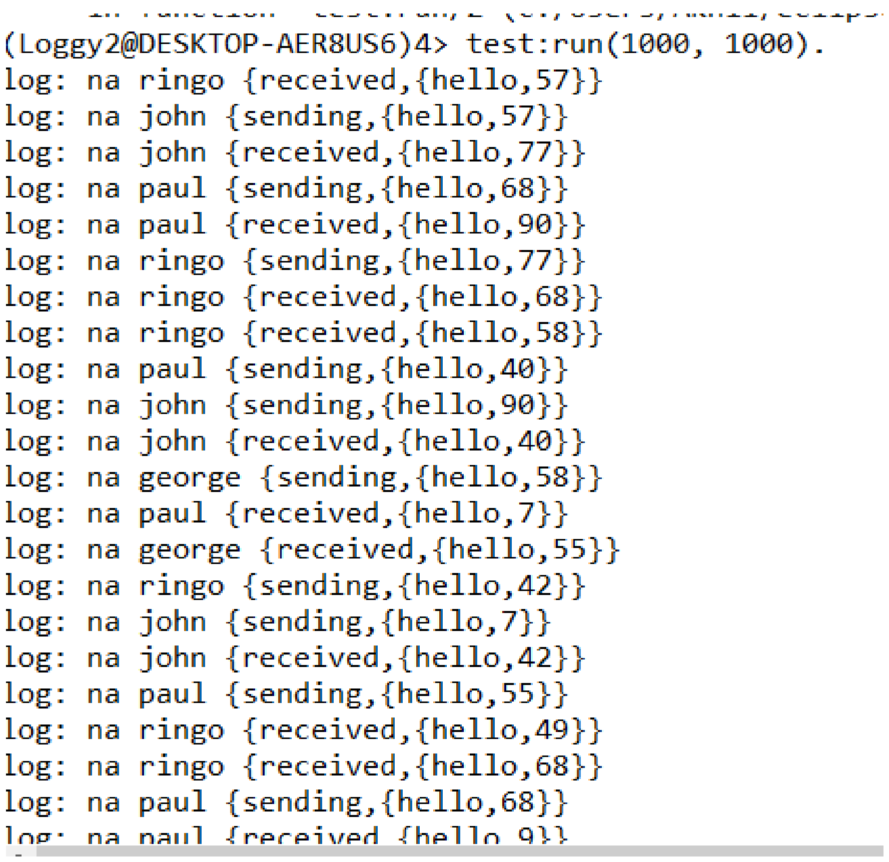
Initially, the worker procedure contains required nodes and their peers respectively. After certain wait period, a message to the respective node will be sent along with the incremented counter. A log shall be made every time when a process sends and received a message along the Lamport time stamp. We use the logger to log, order and add the messages to the queue. In the end, we verify if we can safely log the event. We use a test code to set the nodes, introduce jitter and sleep time and also determine the peers for each of the worker processes.

Main problems and solutions

The problems involved in this is creation of a procedure to order the messages. Here we implement a process to store the messages in a queue. Each worker has a list of messages that contains the logical timestamp and the contents of the message. When a message is received, it is saved into the list associated to the worker that sent the message. After, we look inside all the list of workers, If the list contains at least one message, we can display the message. We look at the message that have the smaller timestamp and display it.

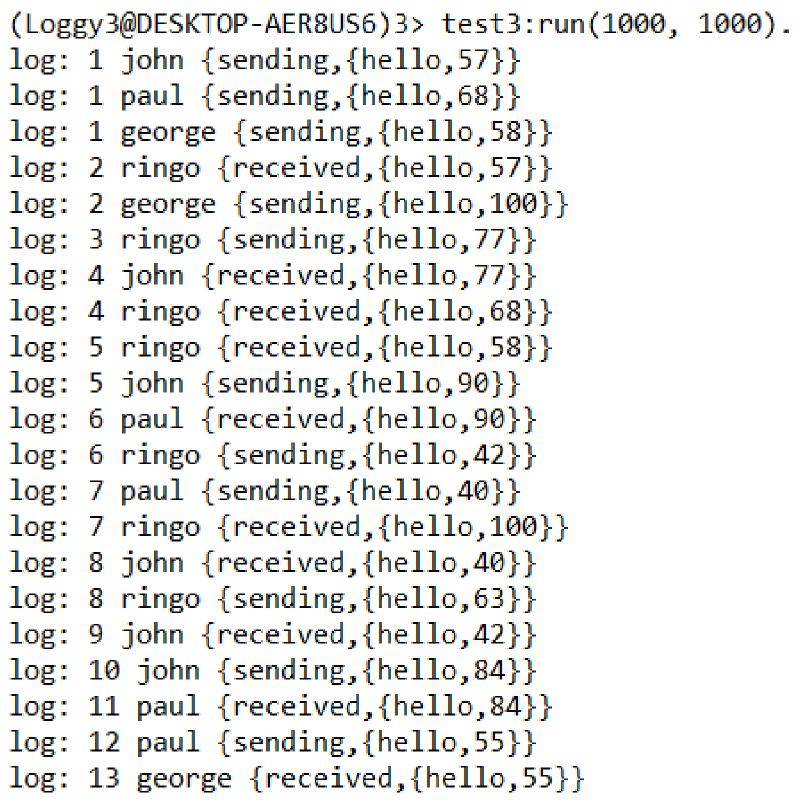
Evaluation

1) In this case we can observe that the received messages display ahead of the sending messages. Here, we haven't yet implemented the time module (Lamport clock). We just logged the messages here.



```
(Loggy2@DESKTOP-AER8US6)4> test:run(1000, 1000).
log: na ringo {received,{hello,57}}
log: na john {sending,{hello,57}}
log: na john {received,{hello,77}}
log: na paul {sending,{hello,68}}
log: na paul {received,{hello,90}}
log: na ringo {sending,{hello,77}}
log: na ringo {received,{hello,68}}
log: na ringo {received,{hello,58}}
log: na paul {sending,{hello,40}}
log: na john {sending,{hello,90}}
log: na john {received,{hello,40}}
log: na george {sending,{hello,58}}
log: na paul {received,{hello,7}}
log: na george {received,{hello,55}}
log: na ringo {sending,{hello,42}}
log: na john {sending,{hello,7}}
log: na john {received,{hello,42}}
log: na paul {sending,{hello,55}}
log: na ringo {received,{hello,49}}
log: na ringo {received,{hello,68}}
log: na paul {sending,{hello,68}}
log: na paul {received,{hello,9}}
```

2) In this case we can observe that the messages are being displayed in order with a Lamport time stamp.



```
(Loggy3@DESKTOP-AER8US6)3> test3:run(1000, 1000).
log: 1 john {sending,{hello,57}}
log: 1 paul {sending,{hello,68}}
log: 1 george {sending,{hello,58}}
log: 2 ringo {received,{hello,57}}
log: 2 george {sending,{hello,100}}
log: 3 ringo {sending,{hello,77}}
log: 4 john {received,{hello,77}}
log: 4 ringo {received,{hello,68}}
log: 5 ringo {received,{hello,58}}
log: 5 john {sending,{hello,90}}
log: 6 paul {received,{hello,90}}
log: 6 ringo {sending,{hello,42}}
log: 7 paul {sending,{hello,40}}
log: 7 ringo {received,{hello,100}}
log: 8 john {received,{hello,40}}
log: 8 ringo {sending,{hello,63}}
log: 9 john {received,{hello,42}}
log: 10 john {sending,{hello,84}}
log: 11 paul {received,{hello,84}}
log: 12 paul {sending,{hello,55}}
log: 13 george {received,{hello,55}}
```

Conclusion

Overall, there has been significant learning in understanding the logging mechanism. The usage of logical time by a process to mark its events and how the ordering mechanism takes place was understood very clearly.