Homework 4: Groupy

Patrik Zhong

October 5, 2021

1 Introduction

In this homework assignment we examined the issues with grouping together processes and syncing them. In this particular case, we try to sync together flashing lights that appear as consoles. In this assignment we iteratively build up a solution that solves problems that arises with crashing nodes.

2 GMS1

In this section we create the most basic of syncronization, where every node simply listens to their master. The master broadcasts to each slave and each respective slave sends out their own message (in the form of a colour). This works quite well, and can even handle crashes with one big notable exception. If the leader dies, everyone will just stop broadcasting their color. This makes a lot of sense, since we mentioned before, the leader has single control of the broadcasting.

3 GMS2

In this particular module we introduce, we introduce elections, but not very democratic ones. If a leader dies, we hold an election that picks the next leader who's simply the next one in our list of nodes, thus bypassing the issue of our programs completely stopping. We see that if we manually close a window, things keep chugging along. But to furhter complicate things, a random crash is introduced into our module. This random crash seemingly desyncs our program every time one node dies. Why?

This is because how we handle thing in our broadcast. When our leader in the broadcast dies, we could've been in the middle of the list. If it crashes, a message is missed and they become out of sync.

4 GMS3

This issue is thus solved in the 3rd module, we're we implement counters and a history where each node remembers their last message. The counters are used to check whether a message is new or old, so if a leader crashes and the next leader resends, each node will know whether to send a message if it's new, or just continue

if the message is old. We also make sure that each node remembers the last message, so that they also know what to resend if a crash happens.

5 GMS4

In this section, we discuss the 3 main issues that are presented to us. The first being what to do if we want to know if we lose a message! The answer to this is that we utilize a TCP/IP type of solution. We simply implement acknowledgements, where the leader sends out an ackrequest and the nodes have to respond along with a counter that shows it has acknowledges the correct message. The issue with this is the extra overheard of the traffic.

The second case being how the monitor function in erlang isn't truly perfect. Several things can happen, such as if a program is stuck in an infinite loop, the monitor will still detect a heartbeat and things are ok and that progress is happening (basically the halting problem), or that the leader is lagging like for network reasons, and that the monitor things it's down even when it's not. We can implement timeout's to solve this.

The third issue is how we could send faulty messages and such even if everything else is correct. The only scenario I could think of is if we have a lagging leader. Since the leader lags out, we think it's dead and hold an election for a new leader. But then the old leader reconnects, and we're stuck with two leaders. We can solve this with ack's or simply that we throw away any old leader by checking for that particular case.

6 Problems and Testing

The main issues I faced in this task was the conceptual parts. Since most of the code was given, it was up to decode and figure out what each part had as a role. The first issue I faced was the sheer confusing of starting up the first test and having a blinking screen on you. Since I couldn't add things due to a bug, I simply had a blinking screen that was admitedly very pretty, but confusing. A lot of given code also required a lot of modification, especially in GMS3 where the main biggest issue of this code lied wherein, figuring out whether a function needed to pass a N+1 or N.

When it came to testing, the test file was mainly used. Using the example, we spawned a worker and then added nodes to it. Additionally we also tested if we could add to the process to keep it from

7 Conclusion

This assignment was one of the very interesting assignments we solved. Seeing how you can solve the issue of programs crashing and implementing workaround and solutions really shed light on how stable certain concurrent processes can be.