Philosophers

Lucas Larsson

Spring Term 2022

Introduction

It is a prerequisite that the reader has read the assignment instructions so i will jump right into it, main goals of this assignment is to get working on concurrency and see how effective it is and what problems does it solve.

It is used here so solve the problem of having multiple philosophers eating at the same time with shared utensils. The classical description is five philosophers are at a round table and to eat noodles(like real students :)) and to the left of each of them a single chopstick, as noodles needs two chopsticks a maximum of 2 philosophers can eat tat the same time. And this is the problem to be solved, how can we feed all of them so non of them starve to death? .

Method

Method used to solve this this assignment is through using the resource below:

The assignment instructions: most helpful thing one can do is to read and reread over the instructions, often i get stuck and find the solution to my problems there, when ever i feel like i don't know what the next step i go back to them.

Good old Google: not a whole lot of help is provided for elixir due to it's popularity, but the philosophers problem is widely covered and there is a lot of similar solutions in other languages to take inspiration from. This method combined by Elixir documentation is a guaranteed way to success.

Professors GitHub repository: this is a last resort kind of method, if i followed the above steps and still didn't solve the problem i check it, the reason for it being a last resort is that i find it hard to think on my own and come up with an original solution once i saw actual solution.

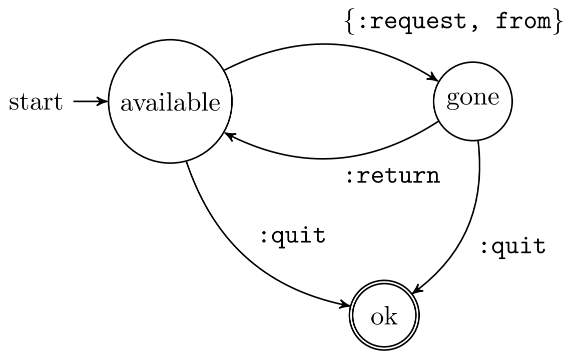
Solution

A chopstick

Chopsticks are either being used by a philosopher `gone`, or free to get picked up by a philosopher `available`. It always starts as available though.

By default process are isolated and independent (Elixir Documentation), but if we want to know and act depending on the status of a process we link them using `spawn_link/1`.

Following the skeleton code example i implemented the methods `start/0, available/0, gone/0` and `request/2`. I even learned a bit about `defp` i.e using private functions so that the methods are only accessible from inside the module. The figure below is taken from the instructions and it shows the chopstick process live cycle.



A philosopher

Philosophers are the core of the program hence the name of the assignment, similar to chopstick they have 2 states. `dreaming/thinking` or `eating`. My first approach to the solution was to just set multiple timers that are triggered by the ending of a each other. Which turned out to be somewhat the right solution.

I used the same `sleep/1` methods suggested with the function `:rand.uniform/1`, next step following the instructions I implemented the philosophers method `start/5` with the 5 requested parameters.

After that i started thinking of the flow that the philosophers follow, they think, eat and repeat basically. The sub flow is that a philosopher wakes up from thinking/dreaming then they grab two chopsticks, eat and go back to dreaming/thinking. If a Philosopher wakes up and try to eat but can't because the chopsticks are used by another philosopher they wait a bit and then go to dreaming state again, if this is repeated multiple times and the `Strength` level gets to 0 the process dies. (see below)

```
defp dreaming( 0,hunger, _left, _right, name, ctrl) do
    IO.puts("#{name} has starved to death :(")
    send(ctrl, :done)
end
```

`start/5` kick start the philosopher with a single `spawn_link/1` function to track the process

Dinner

`start/0, init/0` and `wait/2` methods are defined according to the instructions, so no code is included here or further explanation, the Dinner module kick start all the processes, monitor them and make sure to terminate them if needed.

Experiments

It took me a lot of experimenting to get one round of dinner where all philosophers ate and no one died, I kept running into deadlocks or some of the philosophers would die, it wasn't until i set constant times for the philosophers, could I get it to work.

Asynchronous requests

This is one of the most obvious solutions that can speed up the process, as expressed in the instructions instead of requesting one chopstick at a time, and wait for the respond until you get the second, one could request both chopsticks and wait for them both to get granted. To do this a `granted` function was added.

Answering the question regarding the identification of the chopsticks. It is easy, all of them are process, from the `processes` module Elixir "We can retrieve the PID of a process by calling `self/0`". And then send it back with the return. But I don't see why or how would that help, please let me know if i am missing something.

A waiter

This is another solution like the Asynchronous requests derived from real world application, one usually have a waiter to organize ones meal and bring all the necessary resources i.e chopsticks in this case. A waiter would be great here when optimized for something like only allowing 2 philosophers to eat a t a time, and organising a priority queue. But as a general solution it doesn't feel like a good approach since a waiter needs to have more general information in order to prioritise correctly, and this moves it from a deadlock to a scheduling problem.

Reflection

This is a bit of a different layout compared to my previous reports and it is due to feedback i got where i don't explain enough about the program. So almost no code was included to keep the report short and just explain and discuss the solution. A note regarding this assignment is that debugging code was a nightmare and the bench marks were never consistent, other than that i think it has been an interesting assignment and a nice light introduction to concurrent programming, deadlocks and other stuff to look out for and by mindful of.