FAF.PTR16.1 -- Project 1

Performed by: Covtun Serghei, group FAF-203
Verified by: asist. univ. Alexandru Osadcenco

P0W1

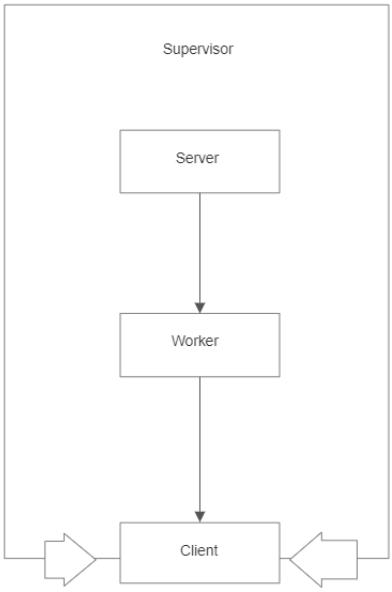
Task1: Write an actor that would read SSE streams. The SSE streams for this lab are available on Docker Hub at alexburlacu/rtp-server, courtesy of our beloved FAFer Alex Burlacu. Create an actor that would print on the screen the tweets it receives from the SSE Reader. You can only print the text of the tweet to save on screen space. Create a second Reader actor that will consume the second stream provided by the Docker image. Send the tweets to the same Printer actor. Continue your Printer actor. Simulate some load on the actor by sleeping every time a tweet is received. Suggested time of sleep – 5ms to 50ms. Consider using Poisson distribution. Sleep values / distribution parameters need to be parameterizable. Create an actor that would print out every 5 seconds the most popular hashtag in the last 5 seconds. Consider adding other analytics about the stream.

```elixir
defmodule Supervisorreader1 do
  use Supervisor

  def start_link() do
    Supervisor.start_link(__MODULE__, [], name: __MODULE__)
    |> elem(1)
  end

  def init([]) do
    children = [%{
        id: :r1,
        start: {Reader1, :start_link, ["http://localhost:4000/tweets/1"]}}, %{
        id: :r2,
        start: {Reader1, :start_link, ["http://localhost:4000/tweets/2"]}}]
    Supervisor.init(children, strategy: :one_for_one)
  end
end

```

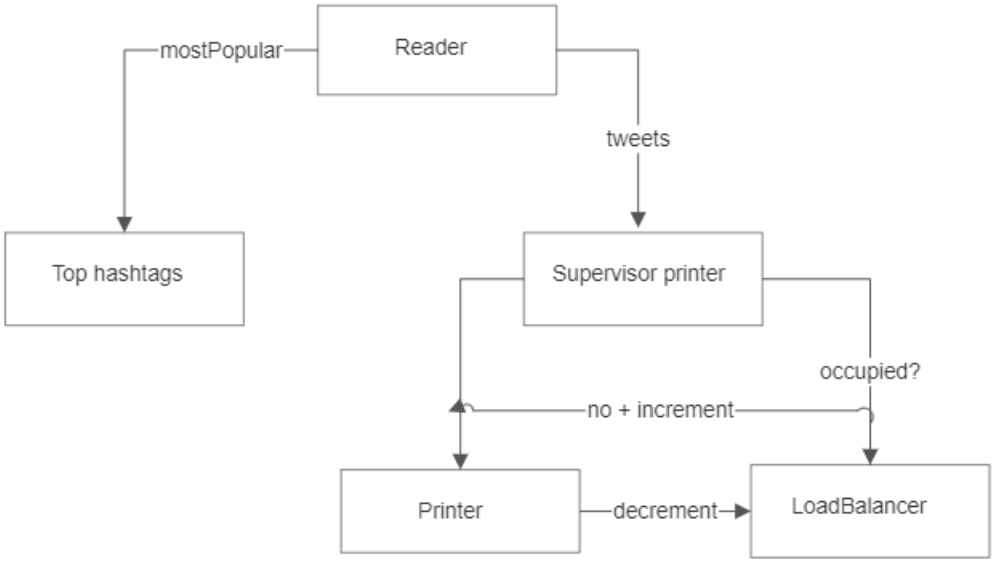
```
@spec start_link :: true
def start_link() do
  process_identifier = Supervisor.start_link(__MODULE__, nr_of_workers: 5, name: __MODULE__)
  |> elem(1)
  Process.register(process_identifier, __MODULE__)
end

@spec init(nil | maybe_improper_list | map) :: {:ok, {map, list}}
def init(args) do
  children = Enum.map(1..args[:nr_of_workers],
  fn i -> %{
      id: i,
      start: {Printer1, :start_link, [i]}}
  end)
  children = children ++ [%{
      id: :b1,
      start: {Balancer1, :start_link, []} }, %{
      id: :m1,
      start: {Mostpopular1, :start_link, []}}]
  Supervisor.init(children, strategy: :one_for_one)
end
```

```
random_time_for_sleep = :rand.uniform(45) + 5
Process.sleep(random_time_for_sleep)
IO.inspect(self())
IO.inspect("Text: \n#{text} \n~FLAG:END_OF_TEXT")
Balancer1.decrement(state)
{:noreply, state}
```

```
def handle_cast(hashtag, state) do
  hashtags = state
  |> elem(1)
  hashtags = Map.update(hashtags, hashtag, 1, fn x -> x + 1 end)
  time_starting = state
  |> elem(0)
  time_ending = :os.timestamp()
  {_, ending_seconds, ending_microseconds} = time_ending
  {_, starting_seconds, starting_microseconds} = time_starting
  time_passed = (ending_seconds - starting_seconds) + (ending_microseconds - starting_microseconds) / 1000000 # 1000000 microsecons = 1 sec
  if(time_passed > 5) do
    hashtags = hashtags
    |> Enum.sort_by(&elem(&1, 1), &>=/2)
    |> Enum.take(5)
    IO.puts("\n\n\t\t ==== Top 5 hashtags in the last 5 seconds ==== \t\t\n\n")
    hashtags
    |> Enum.each(fn {hashtag, count} ->
      IO.puts("#{count} ----- #{hashtag}") end)
    {:noreply, {:os.timestamp(), %{}}}
  else
    {:noreply, {time_starting, hashtags}}
  end
```



Message flow diagram



Supervision Tree Diagram

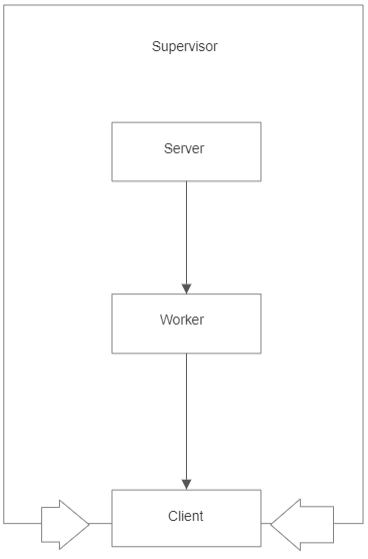
P0W2

Task: Create a Worker Pool to substitute the Printer actor from previous week. The pool will contain 3 copies of the Printer actor which will be supervised by a Pool Supervisor. Use the one-for-one restart policy. Create an actor

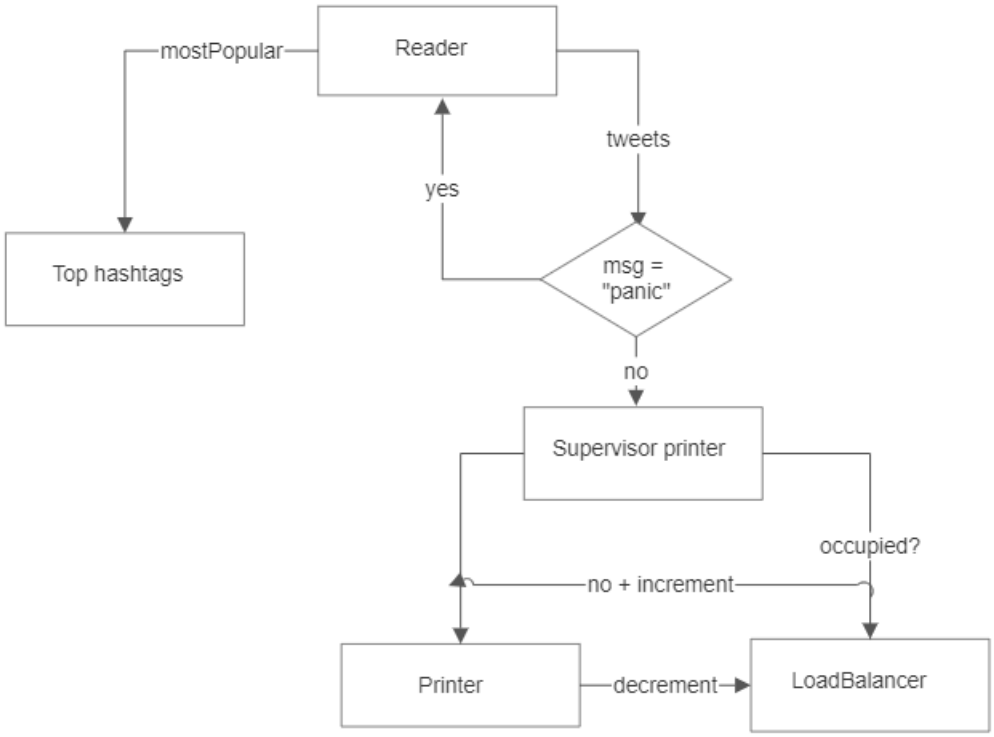
that would mediate the tasks being sent to the Worker Pool. Any tweet that this actor receives will be sent to the Worker Pool in a Round Robin fashion. Direct the Reader actor to sent it's tweets to this actor. Continue your Worker actor. Occasionally, the SSE events will contain a "kill message". Change the actor to crash when such a message is received. Of course, this should trigger the supervisor to restart the crashed actor.

```
if success == :ok do
  if data["message"] == "panic" do
    Supervisorprinter1.tweet_text_printer("panic")
  end
  text = data["message"]["tweet"]["text"]
```

```
def handle_cast(text, state) do
  if text == "panic" do
    IO.puts("\n\n\n\n\n\t\t ==== Killed Printer ==== \t\t\n\n\n\n\n\n")
    Balancer1.decrement(state)
    {:stop, :normal, state}
  else
    random_time_for_sleep = :rand.uniform(45) + 5
    Process.sleep(random_time_for_sleep)
    IO.inspect(self())
    IO.inspect("Text: \n#{text} \n~FLAG:END_OF_TEXT")
    Balancer1.decrement(state)
    {:noreply, state}
  end
end
```



Message flow diagram



Supervision Tree Diagram

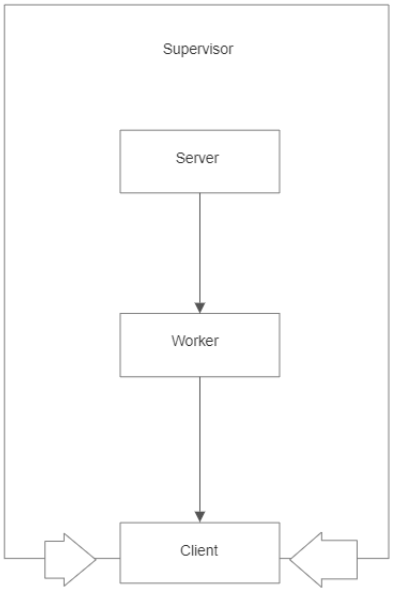
P0W3

Task1: Continue your Worker actor. Any bad words that a tweet might contain mustn't be printed. Instead, a set of stars should appear, the number of which corresponds to the bad word's length. Consult the Internet for a list of bad words.

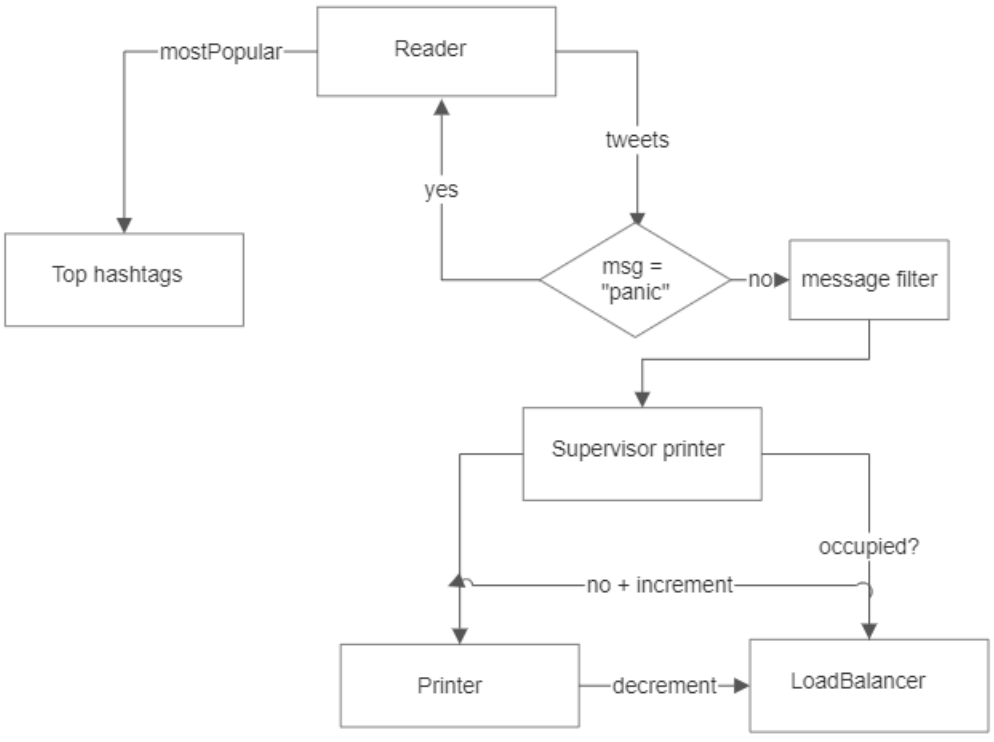
```
bad_words = [
  "arse",
  "arsehole",
  "ass",
  "asshole",
  "balls",
  "bastard",
  "beaver",
  "beef curtains",
  "bellend",
  "bint",
  "bitch",
  "bloodclaat",
  "bloody",
  "bollocks",
```

```
censored_text =
  for word <- String.split(text, " ") do
    if Enum.member?(bad_words, word) do
      String.duplicate("*", String.length(word))
    else
      word
    end
  end
  |> Enum.join(" ")
IO.inspect(censored_text)
```

```
def replace_bad_words(str, bad_words) do
  Enum.reduce(bad_words, str, fn word, acc ->
    String.replace(acc, word, String.duplicate("*", String.length(word)))
  end)
end
```



Message flow diagram



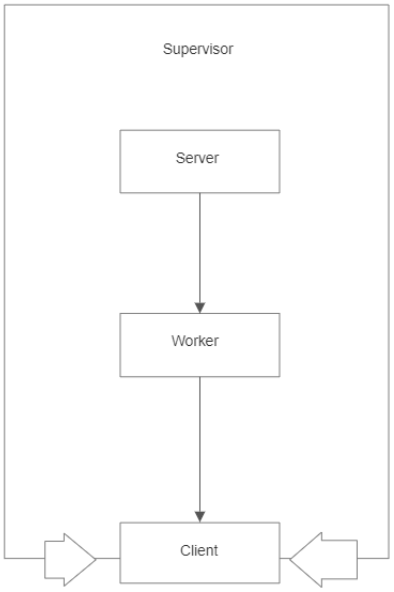
Supervision Tree Diagram

P0W4

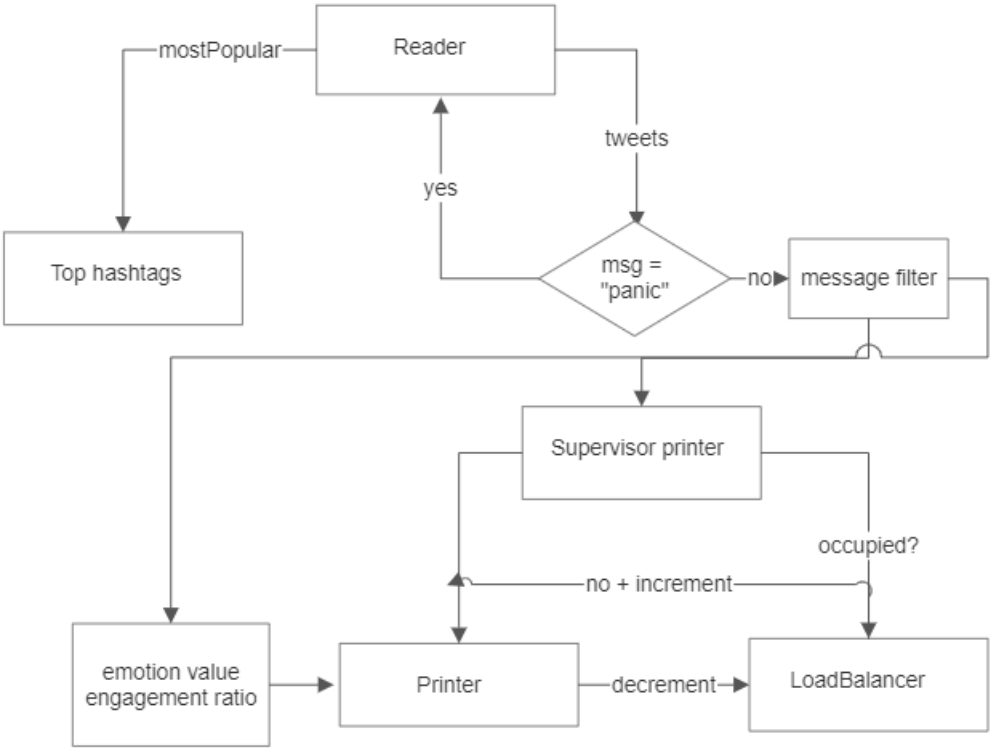
Tast1: Continue your Worker actor. Besides printing out the redacted tweet text, the Worker actor must also calculate two values: the Sentiment Score and the Engagement Ratio of the tweet. To compute the Sentiment Score per tweet you should calculate the mean of emotional scores of each word in the tweet text. A map that links words with their scores is provided as an endpoint in the Docker container. If a word cannot be found in the map, it's emotional score is equal to 0. The Engagement Ratio should be calculated as follows: engagement ratio = #f avourites + #retweets / #followers

```
followers = data["message"]["tweet"]["user"]["followers_count"]
retweet = data["message"]["tweet"]["retweet_count"]
favourites = data["message"]["tweet"]["user"]["favourites_count"]
```

```
if followers != 0 do
  engagement_ratio = (favourites + retweet) / followers
  IO.inspect(engagement_ratio, label: "engagement_ratio")
end

json = File.read!("./emotion.json")
table = Poison.decode!(json)

words = String.split(unsensored_text, " ")
score = Enum.reduce(words, 0, fn word, acc ->
  case Map.get(table, word) do
    nil -> acc + 0
    value -> acc + value
  end
end)
IO.inspect(score, label: "emotion score")
IO.inspect("END OF FLAGG")
```



Message flow diagram



Supervision Tree Diagram

Conclusion.

During this laboratory work we learned more of elixir language, also we tried to write programs in functional way. I learned how to work with SSE streams and also to create stream processing system.

Bibliography.

https://elixir-lang.org/docs.html

https://hexdocs.pm/elixir/1.12.3/Kernel.html

https://elixirschool.com/en