Topic: Stream Processing with Actors

Course: Real Time Programming (PTR)

Author: Tincu Catalin FAF-201

P1W1

Minimal: Initialized a VCS repository for my project. Minimal: Wrote an actor that would read SSE streams. I take id for killing the Actor that receive an alert message and tweetText to print the text of the tweet.

```
val ReaderActor = Source
  .single(Uri("http://localhost:4000/tweets/1"))
  .map(uri => HttpRequest(uri = uri))
  .mapAsync(1)(Http().singleRequest(_))
  .flatMapConcat(_.entity.dataBytes)
  .map(_.decodeString("UTF-8"))
  .map(ServerSentEvent(_))
  .map(e => (System.nanoTime(), e.data))
  .runForeach {
    case (id, tweetText) =>
      // Send tweet text and id to the printer actor pool
      printerActorPool ! PrinterActorPool.Print(id, tweetText)
  }
```

Minimal: Actor that will print on the screen only the text of the tweets. This is done with PrinterActor using Regex, I am finding the tweetText in tweetJson. I have in Printer Actor 2 cases, one when I find some text, and second when I don't and I need to kill the Actor, but that part will be presented lateron.

```
val tweetRegex: Regex = """(?s).*"text":"(.*?)".*""".r
        tweetRegex.findFirstMatchIn(tweetJson) match {
          case Some(m) =>
            val tweetText = m.group(1).replaceAll("\\\\/", "/")
```

Main: Delay for Printer Actor

```
val printerBackoffOptions = Backoff.onFailure(
  Props[PrinterActor],
  "PrinterActor",
  1.second,
  30.seconds,
  0.2
).withManualReset.withSupervisorStrategy(printerSupervisorStrategy)
```

P1W2

Minimal: Worker Pool supervised with Pool Supervisor. One-for-one restart policy.

```
  val printerSupervisorStrategy = OneForOneStrategy() {
    case _: Exception => SupervisorStrategy.Restart
  }
```

Minimal: Round Robin fashion

```
  val printerActorPool = system.actorOf(
    BackoffSupervisor.props(printerBackoffOptions)
      .withRouter(new RoundRobinPool(3)),
    "PrinterActorPool"
  )
```

Main: Kill message and restarting the Actor. This is the second case when we try to find the text attribute in tweetJson and program doesn't find it, it kills the Actor and BackoffSupervisor restarts it.

```
 var id: Long = 0L
           case None =>
            // Kill the actor and restart it
            context.parent ! BackoffSupervisor.reset
            println(s"\nPrintActor $id was killed")
```

Output

```
PrintActor 107788031683100 was killed
```

P1W3

Minimal: Bad words to be censored with '*'. I searched on internet for a list of Bad words, then made a function that will take as a parameter a String which will be my tweetText extracted from tweetJson and take asterisks and multiple with word length. After that in PrinterActor to replace the BadWord with the output of the function.

```
val badWords = Set("Whore", "whore"...
  val asterisks = "*"

  def replaceBadWords(word: String): String = asterisks * word.length

              val filteredTweet = badWords.foldLeft(tweetText) { (text, word) =>
              text.replaceAll(word, replaceBadWords(word))
```

Output

```
Text: life's too short to tolerate **** that doesn't make you happy.
```

P1W4

Minimal: Sentiment Score. It takes the words from emotion_values, it makes the words from tweetText lowercase to match the words from the link. Requests are done over http and it takes the words if it finds a match it takes the number that is converted to Int in able to sum numbers of each word and provide an accurate result. The words that are not in the list per condition are 0.

```
  def calculateSentimentScore(tweetText: String)(implicit system: ActorSystem): Int = {
    val emotionValuesUrl = "http://localhost:4000/emotion_values"

    val request = HttpRequest(uri = emotionValuesUrl)
    val responseFuture = Http().singleRequest(request)

    val response = Await.result(responseFuture, Duration.Inf)
    val emotionValues = Await.result(Unmarshal(response.entity).to[String], Duration.Inf)

    val wordToValueMap = emotionValues
      .linesIterator
      .map(line => {
        val pair = line.split('\t')
        pair(0).toLowerCase -> pair(1).toInt
      })
      .toMap

    tweetText.split("\\s+")
      .foldLeft(0) { (score, word) =>
        val wordScore = wordToValueMap.getOrElse(word.toLowerCase, 0)
        score + wordScore
      }
  }
}
```

Engagement Ratio. In order to implement this condition I needed to extract the favourites_count, retweet_count and followers_count from the same tweetJson. Then it converts those values to Int and eliminates 0 value. The calculation of engagement ratio will be presented in bonus task, but I will specify the formula here as well ((favourites_count+retweet_count)/followers_count).

```
val favCountRegex: Regex = """(?s).*"favourites_count":(\d+).*""".r
val retweetCountRegex: Regex = """(?s).*"retweet_count":(\d+).*""".r
val followersCountRegex: Regex = """(?s).*"followers_count":(\d+).*""".r

val userFollowersCount = followersCountRegex.findFirstMatchIn(jsonFields).map(_.group(1).to
val tweetFavCount = favCountRegex.findFirstMatchIn(jsonFields).map(_.group(1).toInt).getOrE
val tweetRetweetCount = retweetCountRegex.findFirstMatchIn(jsonFields).map(_.group(1).toInt
```

Output

```
Text: @OoMsBlueEyesoO I seen it a million times. I'm like ........
Sentiment Score: 2
Engagement Ratio: 19.531578
```

Bonus: Engagement Ratio per user. The statistic for each user is displayed every 10 seconds, we extract user name from the same tweetJson. We have 2 cases, one for the tweets that had an Engagement Ratio before, and we do increment the count to specify at the end the number of tweets of each user. Second case being when the program collects Engagement Ratio for the first time. Also for the first case it calculates the avg of Engagement Ratio being the totalRatio divided by the count which represents the number of tweets for a specific user.

```
    override def preStart(): Unit = {
      context.system.scheduler.scheduleWithFixedDelay(10.seconds, 10.seconds, self, Calculat
    }

val userNameRegex: Regex = """(?s).*"user":\{"screen_name":"(.*?)".*""".r

val userName = userNameRegex.findFirstMatchIn(jsonFields).map(_.group(1)).getOrElse("Unknown

            engagementRatios.get(userName) match {
              case Some((count, totalRatio)) =>
                val newCount = count + 1
                val newTotalRatio = totalRatio + engagementRatio
                engagementRatios = engagementRatios + (userName -> (newCount, newTotalRatio)
              case None =>
                engagementRatios = engagementRatios + (userName -> (1, engagementRatio))
            }
      case CalculateEngagementRatio =>
        engagementRatios.foreach { case (userName, (count, totalRatio)) =>
          val avgEngagementRatio = totalRatio / count
          println(s"\nUser: $userName\nNumber of Tweets: $count\nAverage Engagement Ratio: $
        }
        engagementRatios = Map.empty[String, (Int, Float)]
```

Output

```
User: IkemenHealing
Number of Tweets: 1
Average Engagement Ratio: 0.18181819

User: 97karadayi
Number of Tweets: 2
Average Engagement Ratio: 0.052887697
```

```
User: Mikamikachin3
Number of Tweets: 1
Average Engagement Ratio: 0.0
```