Programming Paradigms – Erlang Homework

Elisa Marengo Pietro Galliani

1st Semester 2020/21

The Government of the City of Erlanger, Kentucky, US is conducting a census of the city population. However, there was no guideline on how to collect and store the data and so the government ended up with a single big unorganized file containing the census data. Each line in the file contains information about the age, gender, marital status, occupation, and yearly income of a citizen, separated by a space. However, the order of the information in each line is random. For example, the file may contain lines such as:

Doctor 35 Married $300,000 Female
16 $6,000 Student Male Single

You are tasked by the Erlanger government to realize a concurrent Erlang program to summarize the data. The program consists of three types of processes:

1. One **reader**: it reads the file, and dispatches each line to one of the multiply available scanners; in addition, it is the main orchestrator of the entire program, that is, it spawns the other processes and coordinates their termination.
2. Many **scanners**: each scanner receives lines of text; for each received line, the scanner parses the line and classifies the information in the right category (i.e., age, gender, marital status, occupation, or yearly income). Then it sends each piece of data to the summarizer.
3. One **summarizer**: it receives data (a piece of information from the list of data, e.g. a gender or an occupation) from the various scanners, and keeps track of the census data under the following categorization:
   (a) age in the interval of 0-6, 7-12, 13-18, 19-24, 25-30, 31-45, 45-60, and 61+,
   (b) gender in three categories: Male, Female, and Other,
   (c) marital status in four categories: Single, Married, Divorced, and Widowed,
   (d) occupation under all categories found in the data file
   (e) yearly income in the interval of <$10,000, $10,000-$25,000, $25,001-$50,000, $50,001-$100,000, $100,001-$250,000, and >$250,000.

Implementation Details.

The reader is generated by spawning the function `word_count` which has at least two parameters: `FileName`, that is the name of the input file to be processed, and `N`, that is the number of scanners to be created (this determines the degree of concurrency in the overall program).

The flow of the reader is as follows:

1. If an `initialization` message is received, then the reader
   (a) Spawns a summarizer.
   (b) Spawns `N` scanners.
   (c) Communicates the PID of the summarizer to each scanner.
   (d) Reads the file pointed by `FileName`. It then dispatches each line to a scanner, with a rotating policy: the first line goes to the first scanner, the second line goes to the second scanner, ..., the `N`-th line goes to the `N`-th scanner, the `N`+1-line goes to the first scanner, and so on.
   (e) When the end of file is reached, the reader informs each scanner that no more lines will be sent.
2. The reader waits for the completion of all scanners
3. The reader informs the summarizer about the completion of the work once all scanners have finished
4. The reader closes the file

The flow of each scanner is as follows:

1. The scanner waits for the PID of the summarizer.
2. The scanner cyclically waits for a message.
   (a) If the message is about a line of text, the scanner splits the line into a list of data, classifies the data in the different categories and then send both the category and the value to the summarizer.
   (b) If the message is about the end of file reached, the scanner informs the reader that it has completed its execution.

The flow of the summarizer is as follows:

1. The summarizer cyclically waits for a message.
   (a) If the message is about a kind of data, then the summarizer increments the counter relative to that kind. For example, if the message contains a "Male", then the counter for gender "Male" should be incremented, and if the message contains a "Doctor", then the counter for occupation "Doctor" should be incremented, etc.
   (b) The summarizer also receives a message of completion by the reader. When this message is received, the summarizer prints to the standard output the counters it keeps track of.

In this assignment, in order for the above process to work as described, you can define as many additional functions as you need. You are also expected to define the parameters of the functions above as needed (i.e., it is likely the case

that more parameters than those described above would be needed in order for your solution to work).

Hints.

- To scan a file you can implement the following function which returns a list, where each element is a line of the file
  ```
  readFile(FileName) ->
      {ok, Data} = file:read_file(FileName),
      binary:split(Data, [<<"\n">>], [global]).
  ```
- To parse the information contained in a line `Line` you can first convert the parsed line using
  ```
  binary:bin_to_list(Line)
  ```
  and then, split the line in order to extract the different pieces of information, using:
  ```
  string:tokens(Line, " ").
  ```
  which splits a string in correspondence of a white space and returns a list of the extracted elements. You can assume that the lines in the file are such that each piece of information is separated by a space.
- To close a file you can implement the following function
  ```
  closeFile(FileName) -> file:close(FileName).
  ```
- If you need to convert a string into an integer, you can use
  ```
  string:to_integer(H)
  ```
  which returns two possible tuples: {`Number,[]`} if it succeeds in converting the string into a number `Number`, and {`error,_`} otherwise.

Testing.

Test the program with the file `data.txt` provided on OLE. The result printed by the summarizer should have the following form:

```
Gender
"Female": 49198
"Male": 49129
"Other": 1673

Marital status:
"Married": 47060
"Single": 36702
"Divorced": 8154
"Widowed": 8084

Occupation:
"Professor": 1014
"Pilot": 1065
"Artist": 621
"Chemist": 1111
```

```
"Musician": 604
"Janitor": 705
"Firefighter": 1175
"Lawyer": 1122
...

Age:
0-6: 6028
7-12: 6024
13-18: 5874
...

Yearly income:
<$10,000: 47882
$10,000 - $25,000: 10565
...
```

Submission.

Together with the solution, submit the sequence of commands that you executed to test your program and that solves the problem described in the assignment. You can submit the list in a separate `txt` file or you can write it as comments in the same file of your solution.

Suggestion.

In solving this assignment use an incremental approach where you start implementing some components, test them (e.g., simply printing some debug message) and then move to implement some other parts only after you are sure of the correctness of the previous part.

For instance, you can start implementing the basic processes, which just exchange some message. Then you can move to the parting of the file and of the information contained in it, and you can test that you extract them correctly and handle them in the right way. Then, you can work on the computation of the statistics that you have to print at the end.

Deadline: Thursday, 24.12.20, 07:00 (on OLE)