Walter Cazzola

Home Page
ADAPT Lab.
Curriculum Vitae
Research Topic

Exam of Programming Languages

15 December 2014

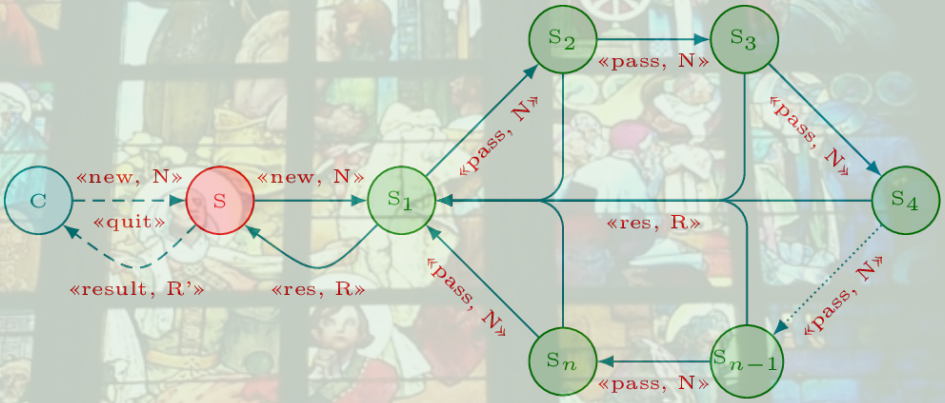
Disclaimer. Note that to have a running solution for an exercise is not enough: you need a well-cooked solution that proves your ability to use what explained during the classes. All the exercises have the same value: 11; Exercise are valued separately and it is necessary to get at least 6 points each exercise to pass the exams, i.e., 18 achieved with only 2 exercises means to fail the exam.

Exercise Erlang: Distributed Sieves.

In the literature you can find a quite interesting actor-based implementation of the sieve of Eratosthenes (for those that do not know what the sieve of Eratosthenes is, it is a quite traditional algorithm to determine if a number is prime or not).

Basically a pool of consecutive prime numbers are used as sieves and the number to be tested will pass or not through a sieve depending on the fact it is divisible or not by the corresponding prime number; a number blocked by at least one sieve is not prime. Of course the test can be considered valid only for a certain range of values (i.e., from 2 to the square of the highest sieve) when the pool of sieves cannot grow. Each sieve is realized by an actor and all of them check the value concurrently.

In our exercise we slightly limit the approach by putting the sieves in an **ordered ring** and passing the value from a sieve to the successive when the value is not blocked and stopping the pass through when proved prime or not prime. The following diagram shows how the actors communicate (the set of messages displayed in the picture **MUST** be implemented, doesn't matter if it is possible to have a better solution this is the requested set of messages to share).



In the scenario we have a remote client that asks a server for checking if a number is prime or not. The server is the controller for the group of sieves and will communicate always with the same sieve and will receive the answer only from such a sieve (that could be considered the *first* in the ring and reasonably this is the lowest one). All the sieves are connected to the successive in the ring and with the *first* and pass to the next the value to check when they can't block it or return a result to the *first* sieve when they determine that the value is prime or not.

Note. all the sieves are implemented in the same way and behave similarly without any exceptions. If the value is too large the controller returns an error message to the client. The controller and the ring of sieves are started by the `start` function in the `controller` module; the parameter represents the range where the sieves are on (that is, no sieve can be higher than the parameter and all the primes in the range from 1 to the parameter are sieves in the ring). Implement three modules: `client`, `controller` and `sieve`.

The following is an example of the expected behavior (an empty line separates the server from the client shell):

```
[20:30]cazzola@surtur:~/lp/erlang>erl -sname sif
(sif@surtur)1> controller:start(100).
You asked for: 25
You asked for: 7
You asked for: 1969
You asked for: 9408
You asked for: 9409
You asked for: 43
You asked for: 430
I'm closing ...

[20:34]cazzola@surtur:~/lp/erlang>erl -sname amora
(amora@surtur)1> client:is_prime(25).
```

```
"is 25 prime? false"
(amora@surtur)2> client:is_prime(7).
"is 7 prime? true"
(amora@surtur)3> client:is_prime(1969).
"is 1969 prime? false"
(amora@surtur)4> client:is_prime(9408).
"is 9408 prime? false"
(amora@surtur)5> client:is_prime(9409).
"9409 is uncheckable, too big value."
(amora@surtur)6> client:is_prime(43).
"is 43 prime? true"
(amora@surtur)7> client:is_prime(430).
"is 430 prime? false"
(amora@surtur)8> client:close().
"The service is closed!!!"
```

All the solutions not conform to the specification will be considered wrong.