My favorite Erlang Program [1]

Joe Armstrong

November 21, 2013

[1] This version of Joe's blog post was translated, edited and annotated by Eric Bailey on January 14, 2017.

Joe Armstrong. My favorite erlang program. https://joearms.github.io/2013/11/21/My-favorite-erlang-program.html, November 2013

The other day I got a mail from Dean Galvin from Rowan University. Dean was doing an Erlang project so he asked "What example program would best exemplify Erlang".

He wanted a small program, that would be suitable for a ten minute talk that would best show off the language. I thought for a while ... and quickly wrote my favorite program, it's the "Universal server".

The Universal Server

Normally servers do something. An HTTP server responds to HTTP requests, an FTP server response to FTP requests and so on. But what about a *Universal Server*? Surely we can generalize the idea of a server and make a universal server, which we can later tell to become a specific server.

Here's my universal server:

1 ⟨*The Universal Server 1*⟩≡ (11)

```
universal_server() ->
    ⟨Wait for a ''become F'' message and become an F server 2⟩
    end.
```

Defines:
`universal_server`, used in chunk 7.

2 ⟨*Wait for a ''become F'' message and become an F server 2*⟩≡ (1)

```
receive
    {become, F} ->
        F()
```

A universal server waits for a `{become, F}` message and then becomes an `F` server.

That was pretty easy. Once I have created a universal server, it just sits and waits for a `{become, F}` message and then it becomes an `F` server.

The Factorial Server

A factorial server is a server which waits for an integer and sends back the factorial of an integer. This is mind-bogglingly simple:

3 ⟨*The Factorial Server 3*⟩≡ (11)

```
factorial_server() ->
    ⟨Wait for an integer N and send back factorial(N) 4⟩,
            factorial_server()
    end.

⟨The factorial function 5⟩
```

Defines:
`factorial_server`, used in chunk 8.

4 ⟨*Wait for an integer N and send back factorial(N)* 4⟩≡ (3)

```
receive
    {From, N} ->
        From ! factorial(N)
```

Uses `factorial` 5.

A factorial server simply waits for an integer n and sends back $n!$.

5 ⟨*The factorial function* 5⟩≡ (3)

```
factorial(0) -> 1;
factorial(N) -> N * factorial(N-1).
```

Defines:
`factorial`, used in chunk 4.

The Erlang definition of `factorial/1` bears a striking resemblance to the recurrence relation:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0. \end{cases}$$

Now we're ready to rock and roll...

Putting It All Together

I'll write a little function that creates a universal server and sends it a "become a factorial server" message. Then I'll send it an integer, wait for the response, and print the response:

6 ⟨*Putting It All Together* 6⟩≡ (11)

```
test() ->
    ⟨Create a universal server 7⟩,
    ⟨Send it a "become a factorial server" message 8⟩,
    ⟨Send it an integer 9⟩
    ⟨Wait for the response and print the response 10⟩
    end.
```

Defines:
`test`, used in chunk 11.

7 ⟨*Create a universal server* 7⟩≡ (6)

```
Pid = spawn(fun universal_server/0)
```

Uses `universal_server` 1.

`test/0` creates a universal server, binding its pid to `Pid`;

8 ⟨*Send it a "become a factorial server" message* 8⟩≡ (6)

```
Pid ! {become, fun factorial_server/0}
```

Uses `factorial_server` 3.

... sends `Pid` a "become a factorial server" message;

9 ⟨*Send it an integer* 9⟩≡ (6)

```
Pid ! {self(), 50},
```

... sends `Pid` 50, asking the factorial server to compute and respond with the value of 50!;

10 ⟨*Wait for the response and print the response* 10⟩≡ (6)

```
receive
    X ->
        io:format("~w~n", [X])
```

... waits for the response and prints the response.

All these functions belong to the module `fav1`:

11 ⟨*fav1* 11⟩≡

```
-module(fav1).
-export([test/0]).

⟨Putting It All Together 6⟩

⟨The Universal Server 1⟩

⟨The Factorial Server 3⟩
```

Uses `test` 6.

N.B. `fav1` exports only `test/0`.

□

Now all we have to do is fire up an Erlang shell and run the test program:

```
$ erl

1> c("src/erlang/fav1.erl").
{ok, fav1}
2> fav1:test().
304140932017133780436126081660647688443776415689605120000000000000
ok
```

Alternatively, we can simply compile and evaluate it from the command line:

```
$ erlc src/erlang/fav1.erl
$ erl -noshell -eval "fav1:test()" -s init stop
304140932017133780436126081660647688443776415689605120000000000000
```

Aside

A few years ago when I was at SICS I had access to Planet Lab. Planet Lab is a research network of 9000 computers. Joining Planet Lab is easy, all you have to do is buy a standard PC, connect it to the network and donate its use to the Planet Lab organization. Having donated your machine to the network, you can use all the other machines.

Planet Lab is a real-world test-bed for distributed applications. It currently has 1171 nodes at 562 sites.

What was I going to do with Planet Lab? I didn't have a clue. What I ended up doing was making some scripts to install empty universal Erlang servers on all the Planet Lab machines (pretty much like the code in this article) - then I set up a gossip algorithm to flood the network with `become` messages. Then I had an empty network that in a few seconds would become anything I wanted it to.

About a year later I had to write a paper. One of the disadvantages of being a researcher is that in order to get money you have to write a paper about something or other. The paper is never about what currently interests you at the moment, but rather what the project that financed your research expects to read about.

Well I had my gossip network setup on Planet Lab and I could tell it to become anything, so I told it to become a content distribution network and used a gossip algorithm to make copies of the same file on all machines on the network and wrote a paper about it and everybody was happy.

Chunks

Index

References

Joe Armstrong. My favorite erlang program. https://joearms.github.io/2013/11/21/My-favorite-erlang-program.html, November 2013.