My favorite Erlang Program[1]

Joe Armstrong

November 21, 2013

[1] This noweb version of the original blog post was translated, edited and annotated by Eric Bailey.

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

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

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

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

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.

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

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

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

Uses `factorial` 5.

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

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

5 ⟨*The factorial function* 5⟩≡ (3)

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

Defines:
`factorial`, used in chunk 4.

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.

5

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

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

Uses `universal_server` 1.

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

6

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

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

Uses `factorial_server` 3.

[6] ... sends `Pid` a "become a factorial server" message;

7

9 ⟨*Send it an integer 9*⟩≡ (6)

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

[7] ... sends `Pid` 50, thereby asking the newly-specialized factorial server to compute and respond with the value of 50!;

8

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

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

[8] ... waits for the response and prints the response.

All these functions belong to the module `fav1`, which exports `test/0`:

11 ⟨*fav1 11*⟩≡

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

⟨Putting It All Together 6⟩

⟨The Universal Server 1⟩

⟨The Factorial Server 3⟩
```

Uses `test` 6.

□

Chunks

Index

References

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