Erlang in 11 Minutes

Sequential Erlang 5 examples

Concurrent Erlang 2 examples

Distributed Erlang 1 example

Fault-tolerant Erlang 2 examples

Bit syntax 1 example

Sequential Erlang

Factorial

```
-module(math).
-export([fac/1]).

fac(N) when N > 0 -> N*fac(N-1;
fac(0) -> 1

> math:fac(25).
 15511210043330985984000000
```

Binary Tree Search

```
lookup(Key, {Key, Val,_,_}) -> {ok, Val};
lookup(Key, {Key1,Val,S,B}) when Key < Key1 ->
   lookup(Key, S);
lookup(Key, {Key1, Val, S, B})->
   lookup(Key, B);
lookup(key, nil) ->
   not_found.
```

Sequential Erlang

append

sort

adder

```
append([H|T], L) -> [H|append(T, L)];
append([],    L) -> L.

sort([Pivot|T]) ->
    sort([X||X <- T, X < Pivot]) ++
    [Pivot] ++
     sort([X||X <- T, X >= Pivot]);
sort([]) -> [].

> Adder = fun(N) -> fun(X) -> X + N end end.
#Fun
> G = Adder(10).
#Fun
> G(5).
15
```

Concurrent Erlang

spawn

send

receive

```
Pid = spawn(fun() -> loop(0) end)

Pid ! Message,
.....

receive
    Message1 ->
        Actions1;
    Message2 ->
        Actions2;
        ...
    after Time ->
        TimeOutActions
end
```

Distributed Erlang

```
Pid = spawn(Fun@Node)

alive(Node),
.....

not_alive(Node)
```

Fault-tolerant Erlang

```
...
case (catch foo(A, B)) of
    {abnormal_case1, Y} ->
        ...
    {'EXIT', Opps} ->
        ...
    Val ->
        ...
end,
...

foo(A, B) ->
    ...
    throw({abnormal_case1, ...})
```

Monitor a process

```
...
process_flag(trap_exit, true),
Pid = spawn_link(fun() -> ... end),
receive
    {'EXIT', Pid, Why} ->
    ...
end
```

Bit Syntax

```
-define(IP_VERSION, 4).
-define(IP_MIN_HDR_LEN,5).
DgramSize = size(Dgram),
case Dgram of
 <<?IP_VERSION:4, HLen:4,
  SrvcType:8, TotLen:16, ID:16, Flgs:3,
  FragOff:13, TTL:8, Proto:8, HdrChkSum:16,
  SrcIP:32, DestIP:32, Body/binary>> when
   HLen >= 5, 4*HLen =< DgramSize ->
    OptsLen = 4*(HLen - ?IP_MIN_HDR_LEN),
    <<Opts:OptsLen/binary,Data/binary>> = Body,
  ...
```

This code parses the header and extracts the data from an IP protocol version 4 datagram