Chapter 12

Erlang Processes

Process Communication

```
-module(ping).
-export([run/0, ping/0]).

run() ->
      Pid = spawn(fun ping/0),
      Pid ! self(),
      receive
            pong -> ok
      end.

ping() ->
      receive
            From ! pong
      end.
```

Echo Process

```
-module(echo).
-export([go/0, loop/0]).

go() ->
      Pid = spawn(echo, loop, []),
      Pid ! {self(), hello},
      receive
            {Pid, Msg} -> io:format("~w~n",[Msg])
      end,
      Pid ! stop.

loop() ->
      receive
            {From, Msg} ->       From ! {self(), Msg},
                              loop();
            stop ->true
      end.
```

Another Echo Process

```
go() ->
      register(echo, spawn(echo2, loop, [])),
      echo ! {self(), hello},
      receive
            {_Pid, Msg} -> io:format("~w~n",[Msg])
      end.
loop() ->
      receive
            {From, Msg} ->  From ! {self(), Msg},
                                    loop();
            stop -> true
      end.
```

Timer Process

```
send_after(Time, Msg) ->
      spawn(my_timer, send, [self(),Time,Msg]).

send(Pid, Time, Msg) ->
      receive
      after
          Time ->      Pid ! Msg
      end.

sleep(T) ->
      receive
      after
          T ->  true
      end.
```

Area Functions

```
-module(geometry).
-export([area/1]).

area({rectangle, Width, Height}) -> Width * Height;
area({circle, R})                -> 3.14159 * R * R.
```

Area Function as a Process

```
loop() ->
  receive
    {rectangle, Width, Ht} ->
       io:format("Area of rectangle is ~p~n",[Width*Ht]),
       loop();
    {circle, R} ->
       io:format("Area of circle is ~p~n", [3.14159*R*R]),
       loop();
    Other ->
       io:format("I don't know what a ~p is ~n",[Other]),
       loop()
    end.
```

Client-Server Version

```
rpc(Pid, Request) ->
    Pid ! {self(), Request},
    receive
       {Pid, Response} ->
           Response
    end.
loop() ->
    receive
       {From, {rectangle, Width, Ht}} ->
           From ! {self(), Width * Ht},
           loop();
       {From, {circle, R}} ->
           From !  {self(), 3.14159 * R * R},
           loop();
       {From, Other} ->
           From ! {self(), {error,Other}},
           loop()
    end.
```

Example Sources

Programming Erlang: Software for a Concurrent World, Joe Armstrong.

Erlang Programming, Francesco Cesarini & Simon Thompson.

Erlang and OTP in Action, Martin Logan, Eric Merritt, Richard Carlsson.