A very simple client server in Erlang

February 5, 2008

1 Introduction

By now you should of read and the first 32 pages of the tutorial pdf file from the web page. You shall now develop a very simple client server system in Erlang.

2 A simple server

Remember that Erlang is a functional language, which means that there are no variables but only values. A server generally has some state. Thus we have to use patterns similar to the following:

```
 server(State) ->
  receive
   some_request ->
     ..... do some stuff
     ..... compute the new state NewState
     server(NewState)
  end.
```

You will extend the following very simple server. Your server should have one piece of state, a value that records how many times it has been called.

```
server(State) ->
    receive
        {request,Return_PID} ->
            io:format("Request received. ~w~n",[Return_PID]) ,
            NewState = State + 1,
            Return_PID ! {hit_count,NewState},
            server(NewState)
    end.
```

The client is simpler, it takes as a parameter the server PID, sends the request and prints out the value.

```
client(Server_Address) ->
    Server_Address ! {request, self()},
    receive
        {hit_count,Number} ->
            io:format("Hit count was ~w~n",[Number])
    end.
```

Now put everything in one file and try it!!! (Actually type in the file yourself, you'll learn a lot just by typing the examples in).

```
-module(simple).

-export([server/1,client/1,start/0]).



server(State) ->
    receive
        {request,Return_PID} ->
            io:format("Request received. ~w~n",[Return_PID]) ,
            NewState = State + 1,
            Return_PID ! {hit_count,NewState},
            server(NewState)
    end.

client(Server_Address) ->
    Server_Address ! {request, self()},
    receive
        {hit_count,Number} ->
            io:format("Hit count was ~w~n",[Number])
    end.


start() ->
    Server_PID = spawn(simple,server,[0]),
    spawn(simple,client,[Server_PID]).
```

3 Many clients

The previous example is a bit boring with only one client. The following function will spawn multiple clients:

```
spawn_n(N,Server_PID) ->
    if
        N>0 ->
            spawn(simple,client,[Server_PID]),
        spawn_n(N-1,Server_PID)
    end.
```

4 Hand in questions and files

1. In the simple server

   ```
   server(State) ->
       receive
           {request,Return_PID} ->
               io:format("Request received. ~w~n",[Return_PID]) ,
               NewState = State + 1,
               Return_PID ! {hit_count,NewState},
               server(NewState)
       end.
   ```

   What does the line

   ```
               Return_PID ! {hit_count,NewState},
   ```

   do?

2. In the simple client

   ```
   client(Server_Address) ->
       Server_Address ! {request, self()},
       receive
           {hit_count,Number} ->
               io:format("Hit count was ~w~n",[Number])
       end.
   ```

   What does `self()` do and why is it used in this example?

3. In the simple `start()` function what does

   ```
       spawn(simple,client,[Server_PID]).
   ```

   do?

4. Modify start to take a single parameter which is the number of clients that have to be spawned. Don't forget to modify the `export` directives correctly. Include the complete listing for this program.

5. Now the owner of the server has the right to query the count without incrementing it. Modify the server by adding code for the message

```
{server_owner,Server_PID}
```

In the following place

```
 server(State) ->
    receive
        {request,Return_PID} ->
            io:format("Request received. ~w~n",[Return_PID]) ,
            NewState = State + 1,
            Return_PID ! {hit_count,NewState},
            server(NewState) ;
        {server_owner,Server_PID} ->
             .......
    end.
```

Modify `start` to have a single owner process that constantly queries the server for the state. You should fill in the `....`s and decide on what other messages are passed between the server and the owner. You should print out the value received from the server using `io:format`.

```
 owner(Server_PID) ->
   Server_PID! .... ,
   receive
     { .... } ->
       ..... ,
       io:format(......) ,
       owner(Server_PID)
   end.
```

Provide a complete listing for this file.

6. You are to give the owner the power to reset the server counter to 0. You should modify the server as follows:

```
 server(State) ->
    receive
        {request,Return_PID} ->
            io:format("Request received. ~w~n",[Return_PID]) ,
            NewState = State + 1,
            Return_PID ! {hit_count,NewState},
            server(NewState) ;
        reset ->
           io:format( .... ) ,
           server(....)
    end.
```

again print out some message to say that you've been reset on the screen, but also you'll need to work out what value pass when you do the recursive call to `server`.

7. Now modify the `owner` process to reset the server counter if it exceeds 100. Look at how the erlang `if` statement works. Hand in the complete listing for this example.