Peer-To-Peer Lab 1 - A very simple client-server system in Erlang

July 7, 2012

1 Introduction

By now you should have read the first 28 pages of "Getting Started with Erlang" which is linked from the instruction homepage or can be found here:
`http://www.erlang.org/download/gettingstarted-5.4.pdf`
You shall now develop a very simple client-server system in Erlang.

2 Preparation

1. Download the required files from `http://www.it.uu.se/edu/course/homepage/p2p/st12/labs/client_server/lab1.tar.gz`
2. Unpack the files.
3. Don't bother going through the file `gui.erl`. It's only there to display something for you, it's not part of the course material.

3 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:

```
1 server(State) ->
2     receive
3         some_request ->
4         %%..... do some stuff
5         %%..... compute the new state NewState
6         server(NewState)
7     end.
```

You will extend the following very simple chat server, similiar to the IRC chat, but only with local message passing. Your server have a piece of state, a list of online nodes in the chat.

```
notify_clients([], _) ->
    ok;
notify_clients([{Pid, _Name}| RestNodes], {Msg, From}) ->
    Pid ! {msg, Msg, From},
    notify_clients(RestNodes, {Msg, From});
notify_clients([{Pid, _Name}| RestNodes], Msg) ->
    Pid ! {online_nodes, Msg},
    notify_clients(RestNodes, Msg).

server(OnlineNodes) ->
    receive
        {send, Msg, From} ->
            notify_clients(OnlineNodes, {Msg, From}),
            server(OnlineNodes);
        {online, Name, Pid} ->
            NewOnlineNodes = [{Pid, Name}|OnlineNodes],
            notify_clients(NewOnlineNodes, NewOnlineNodes),
            server(NewOnlineNodes);
        {offline, Name} ->
            NewOnlineNodes = lists:keydelete(Name, 2, OnlineNodes),
            notify_clients(NewOnlineNodes, NewOnlineNodes),
            server(NewOnlineNodes)
    end.
```

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

```
client(Server, Name) ->
    Server ! {online, Name, self()},
    gui:start(self(), Name, []),
    client_loop(Server, Name).

client_loop(Server, Name) ->
    receive
        {msg, Msg, From} ->
            gui:display_new_msg(Msg, Name, From),
            client_loop(Server, Name);
        {send_msg, Msg} ->
            Server ! {send, Msg, Name},
            client_loop(Server, Name);
        {online_nodes, OnlineNodes} ->
            gui:update_online_nodes(OnlineNodes, Name),
            client_loop(Server, Name);
        go_offline ->
            Server ! {offline, Name}
    end.
```

Now put everything below in a file named `simple.erl` 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/2, start/0]).

start() ->
    Pid = spawn(simple, server, [[]]),
    spawn(simple, client, [Pid, "east"]),
```

```
7       spawn(simple, client, [Pid, "west"]),
8       spawn(simple, client, [Pid, "poff"]).
9
10  notify_clients([], _) ->
11      ok;
12  notify_clients([{Pid, _Name}| RestNodes], {Msg, From}) ->
13      Pid ! {msg, Msg, From},
14      notify_clients(RestNodes, {Msg, From});
15  notify_clients([{Pid, _Name}| RestNodes], Msg) ->
16      Pid ! {online_nodes, Msg},
17      notify_clients(RestNodes, Msg).
18
19  server(OnlineNodes) ->
20      receive
21          {send, Msg, From} ->
22              notify_clients(OnlineNodes, {Msg, From}),
23              server(OnlineNodes);
24          {online, Name, Pid} ->
25              NewOnlineNodes = [{Pid, Name}|OnlineNodes],
26              notify_clients(NewOnlineNodes, NewOnlineNodes),
27              server(NewOnlineNodes);
28          {offline, Name} ->
29              NewOnlineNodes = lists:keydelete(Name, 2, OnlineNodes),
30              notify_clients(NewOnlineNodes, NewOnlineNodes),
31              server(NewOnlineNodes)
32      end.
33
34  client(Server, Name) ->
35      Server ! {online, Name, self()},
36      gui:start(self(), Name, []),
37      client_loop(Server, Name).
38
39  client_loop(Server, Name) ->
40      receive
41          {msg, Msg, From} ->
42              gui:display_new_msg(Msg, Name, From),
43              client_loop(Server, Name);
44          {send_msg, Msg} ->
45              Server ! {send, Msg, Name},
46              client_loop(Server, Name);
47          {online_nodes, OnlineNodes} ->
48              gui:update_online_nodes(OnlineNodes, Name),
49              client_loop(Server, Name);
50          go_offline ->
51              Server ! {offline, Name}
52      end.
```

To make things simpler a Makefile is provided.

1. Go to the folder where you unpacked the files in the "Preparation" section.
2. Open a command-line.
3. Type in the following: `make test`.

4 Many clients

The current program only have one client in it. To make it more fun we want to add several clients. Add the following code to `simple.erl`:

```
spawn_n(0, _ServerPid) ->
    io:format("Last client spawned~n");
spawn_n(N, ServerPid) ->
    spawn(simple, client, [ServerPid, "client_"++integer_to_list(N)]),
    spawn_n(N-1, ServerPid).
```

Also, change the `start/0` function to the following:

```
start() ->
    Pid = spawn(simple, server, [[]]),
    spawn_n(4, Pid).
```

5 Questions

Answers to questions must be given in a file name `answers.txt` or `answers.pdf` and submitted via the Moodle site.

Question 1: In the simple server `notify_clients/2` function

```
notify_clients([], _) ->
    ok;
notify_clients([{Pid, _Name}| RestNodes], {Msg, From}) ->
    Pid ! {msg, Msg, From},
    notify_clients(RestNodes, {Msg, From});
notify_clients([{Pid, _Name}| RestNodes], Msg) ->
    Pid ! {online_nodes, Msg},
    notify_clients(RestNodes, Msg).

server(OnlineNodes) ->
    receive
        {send, Msg, From} ->
            notify_clients(OnlineNodes, {Msg, From}),
            server(OnlineNodes);
        {online, Name, Pid} ->
            NewOnlineNodes = [{Pid, Name}|OnlineNodes],
            notify_clients(NewOnlineNodes, NewOnlineNodes),
            server(NewOnlineNodes);
        {offline, Name} ->
            NewOnlineNodes = lists:keydelete(Name, 2, OnlineNodes),
            notify_clients(NewOnlineNodes, NewOnlineNodes),
            server(NewOnlineNodes)
    end.
```

What does the line

```
Pid ! {msg, Msg, From}
```

do?

Question 2: In the simple client

```
client(Server, Name) ->
    Server ! {online, Name, self()},
    gui:start(self(), Name, []),
    client_loop(Server, Name).

client_loop(Server, Name) ->
    receive
        {msg, Msg, From} ->
            gui:display_new_msg(Msg, Name, From),
            client_loop(Server, Name);
        {send_msg, Msg} ->
            Server ! {send, Msg, Name},
            client_loop(Server, Name);
        {online_nodes, OnlineNodes} ->
            gui:update_online_nodes(OnlineNodes, Name),
            client_loop(Server, Name);
        go_offline ->
            Server ! {offline, Name}
    end.
```

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

Question 3: In the simple start() function, what do does the line:

```
spawn(simple, client, [Pid, "east"])
```

do?

6 Add functionality to client and server

Start with the `simple.erl` file and add the functionality by following the steps below. Hand in the finished simple.erl file on the Moodle site.

Step 1: Create `start/1` (a new function with the same name but takes *(*1 argument). The argument is supposed to be the number of clients to be spawned.

Step 2: The Makefile make things easier, but also creates some problems. So in order for the `start/1` function to work, the function MUST start like this

```
start([N]) ->
    N_clients = list_to_integer(atom_to_list(N)),
    %% Your code goes here
        .
```

Step 3: To use the new start function, type in the following: `make test2`.
Don't forget to modify the export directives so that `start/1` is exported.

7 Test run

When testing the application, the following functionality should be observed:

1. When a message is inputted in one of the client windows (and enter was pressed). That message should be displayed for all the clients.

2. When a client dies (press the X in the upper right corner) that node should disappear from the other clients online list (the box to the left displays the online nodes).