Report : Rudy - a small web server

Vasileios Giannokostas , vasgia@kth.se

September 26, 2014

1 Introduction

The aim of the seminar was the implementation of a web server application in Erlang. The server is able to receive only a Get request from a client and respond with a message. In the simplest scenario the server responses with the same message to the client. After the seminar we will be able to :

- Understand the structure of a simple server process.
- Understand the usage of the socket API of Erlang.
- Understand the usage of the Interface to TCP/IP sockets [gen_tcp module].
- Understand the basic approach of the HTTP protocol.
- Practice in Erlang was also useful .

This web server is a basic client-server application so a simple distributed systems. Our server hosts a server program sharing the resources with the clients and the clients connect to it. This means that the client doesn't share any of its resources but it request a server function.
The connection between a server and a client is based on the usage of the **HTTP protocol** as an application protocol for distributed information systems and on the gen_tcp module provides functions for communicating with sockets using the **TCP/IP protocol**.

2 Main problems and solutions

The first step was to implement the TCP structure of our server. I created a module that is called server.erl and there I put the given code of the description.In the picture below you can see the code of the server.erl file.

A brief explanation of the code:
*T*he function , that initiates the server , opens a listening socket in the given Port. In line 6 we set a socket that listens in the Port. If we have an incoming connection accept the connection (handler function , line 18) and then the request function receives the request(line 20). After this we call the parser function (http.erl module) , we take the reply (reply function) and we send the reply to the client (line 33).

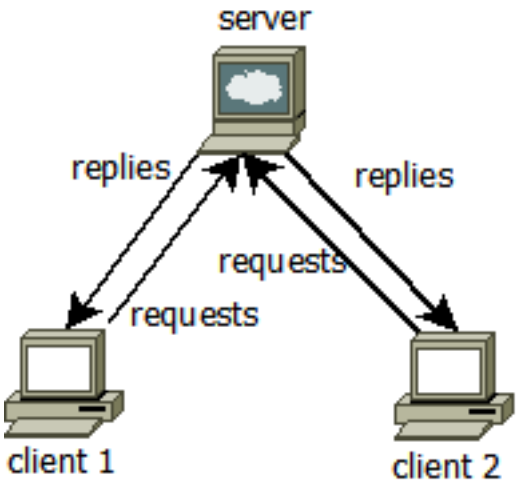
```
1  -module(server).
2  -compile(export_all).
3
4  init(Port) ->
5  Opt = [list, {active, false}, {reuseaddr, true}],
6  case gen_tcp:listen(Port, Opt) of
7  {ok, Listen} ->
8      handler(Listen),
9      gen_tcp:close(Listen),
10     init(Port),
11 ok;
12 {error, Error} ->
13 error
14 end.
15
16
17 handler(Listen) ->
18 case gen_tcp:accept(Listen) of
19 {ok, Client} ->
20     %%request(Client);
21     spawn(fun() -> request(Client) end);
22 {error, Error} ->
23 error
24 end.
25
26
27 request(Client) ->
28 Recv = gen_tcp:recv(Client, 0),
29 case Recv of
30 {ok, Str} ->
31     Req = http:parse_request(Str),
32     Response = reply(Req),
33     gen_tcp:send(Client, Response);
34 {error, Error} ->
35     io:format("rudy: error: ~w~n", [Error])
36 end,
37 gen_tcp:close(Client).
38
39
40 reply({{get, URI, _}, _, _}) ->
41 %timer:sleep(40),
42 http:ok(URI).
43
44 start(Port) ->
45 register(rudy, spawn(fun() -> init(Port) end)).
46
47 stop() ->
48 exit(whereis(rudy), "time to die").
```

figure:server.erl

With this implementation we are able to open a port that it waits for our message. We can also stop this port using the stop function (line 47). I checked it also by using a web browser. The socket listens for my message and replies in every new message , until I stop it.

3 Experiments

We used the given benchmark program to calculate the time it takes to receive the reply. This benchmark program produces many requests and bombards the server with requests. We took measures from the same machine (server and client in the same machine) , also we measures how much time is required for my machine to serve another one (with delay between the replies or no) and finally my machine served two separated machines. I present the results in the tables below.



Now, how many request per second can we serve?

Server/Tester	Delay	Requests/sec
My Machine / My Machine	40ms	≈ 21
My Machine / My Machine	No	≈ 357
My Machine / External Machine	40ms	≈ 16
My Machine / External Machine	No	≈ 67

Is our artificial delay significant or does it disappear in the parsing overhead?

The artificial delay that we insert to the server(for the response) is important and crucially affects the time that is required.The above results clearly depicts this.

What happens if you run the benchmarks on several machines at the same time?

I servered two external machines and I measured this results:

Server/Tester	Delay	Requests/sec
My Machine / External Tester 1	40ms	≈ 5
My Machine / External Tester 2	40ms	≈ 4
Total		≈ 9

Server/Tester	Delay	Requests/sec
My Machine / External Tester 1	No	≈ 6
My Machine / External Tester 2	No	≈ 7
Total		≈ 13

4 Conclusions

In this seminar , we have been taught the basic concept of a http protocol , how the connection between a server and a client established and how to construct a basic TCP server. We connected each other and take measures for the response time. In addition , we were practicing in Erlang and we have learnt another aspect of functional programming.