Rudy - a small web server

Nihad Ibrahimli, nihadi@kth.se

September 6, 2017

Introduction

The assignment was implementing a small web server in Erlang. It contains to implement HTTP parser and using socket API.The goal of this exercise was learning the procedurs for using a socket API, describing the structure of a server process and describing the HTTP protocol.

A HTTP parser

In this part I implemented HTTP get request. In order to impement that it is important to understand the description of the request from RFC 2016. To implement parsing function it should return a tuple consisting of the parsed result and the rest of the string.

The request line consists of a method, a request URI, and a http version. There are some methods OPTIONS, GET, and HEAD, but we are only interesting GET requests.

Next I implemented the parsing of URI. This is done by recursive definition.The URI is returned as a string. Then parsing version and headers is implemented.

I tested HTTP parser like the following lines.

Server

In this part I am expected to start a program that waits for the incoming request and delivers a reply and then terminates. I implemented 4 procedures under rudy module:

- init(Port) initialized the server, takes a port number, opens a listening socket. After the request has been handled the socket will be closed.
- handler(Listen) is listening to the socket for an incoming connecion. Once a client is connected it wil pass the connection to request.
- request(Client) read the request from the client connection and parse it. It will then parse the request using HTTP parser, and the request is passed to reply. The reply is sent back to the client.
- reply(Request): this function defines what to reply.

In order to implement the procedures above gen_tcp library functions are used.

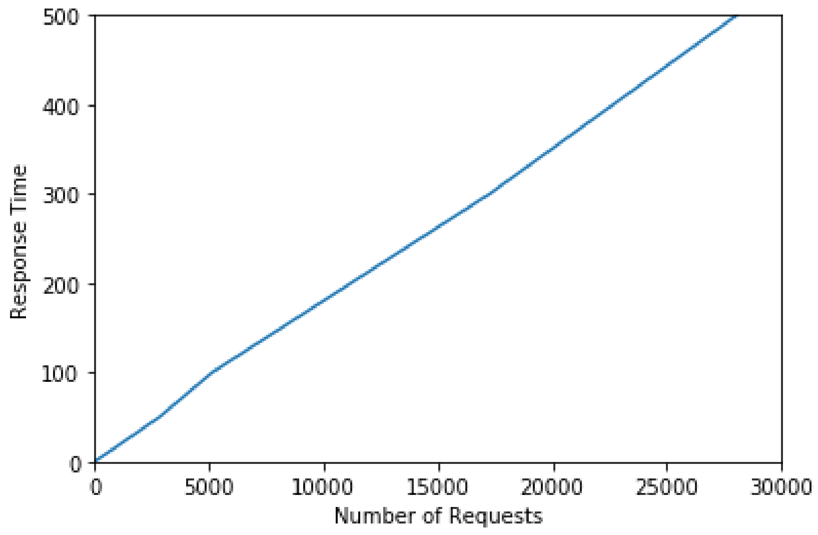
The listening port is opened in init function by gen tcp:listen(Port, Option).

Incoming request will be accepted by gen tcp:accept(Listen), this fnction is included in handler(Listen) which is listening for incoming connections.

Once there is a connection, we read the input and output and return it as a string by gen tcp:recv(Client, 0).

Evaluation

I tested my server with the bench function. Firstly I run it (single thread rudy) with 100 requests. It took 5148 ms for 100 request. So it means 0.05148 second per request. Below you can see in the graph, how the response time increases with the increment of the number of requests.



As seen from the graph the increase is linear.

Then I tried to implement multiprocessor rudy. In order to test test module, bench file also needs to be modified. In the previous version, the request was sent sequentially, however in new version the requests sent in parallel.

Number of Requests	Single process	Multiprocess
50	2839 ms	936 ms
100	5148 ms	1215 ms
300	17269 ms	4679 ms
500	28064 ms	6771 ms

The most interesting testing for me was to compare response time to the requests between different shells in same computer and different computers. Below is the table for single process.

Number of Requests	**Same computer**	**Different computers**
50	2839 ms	3150 ms
100	5148 ms	7046 ms
300	17269 ms	19734 ms
500	28064 ms	31968 ms

Conclusion

Despite the fact most codes were given to us, the task was challenging for me, since I am new to Erlang programming. This task helped me understand how Erlang programs are run. I also learned the functions of gen_tcp library and how to use them.