Report 1: Rudy Server

Mallu Mallu

September 12,2016

1.Introduction

In this assignment a web server “Rudy” is implemented which handles the HTTP requests from client. For parsing the request coming from the client a “parse_request” function has been implemented in erlang which handles the GET requests. Also socket API is used in communication from client to the server and receiving response from the server to client.

In context of distributed systems, I have learnt how sockets communicate using the TCP/IP protocol, basic idea of how server handles the requests coming from clients. I have also learnt some basics of erlang as well.

2. Main problems and solutions

a) Implementing parser_request function in erlang to parse request coming from client: The very first thing we need to implement Rudy is how to parse the request coming from the client and usually for this we considered HTTP GET request only. For parsing the request “parser_request” function is created in “http.erl” file as shown below:

```
parse_request(R0) ->
{Request, R1} = request_line(R0),
{Headers, R2} = headers(R1),
{Body, _} = message_body(R2),
{Request, Headers, Body}.
```

Through this function which calls other functions defined in the same file “request_line”, “headers” and “message_body”. Using erlang pattern matching we fetch request line, header and remaining message body from the coming request respectively.

b) Further request_line is parsed to fetch out Method, Request URI and HTTP version. This is implemented through “request_line” function defined in http.erl.

```
request_line([$G, $E, $T, 32 |R0]) ->
{URI, R1} = request_uri(R0),
{Ver, R2} = http_version(R1),
[13,10|R3] = R2,
{{get, URI, Ver}, R3}.
```

Again this function uses pattern matching and functions “request_uri” and “http_version” to get request URI and http version values respectively.

c)Building Rudy and initializing a port number.

After the http parser is complete we started building Rudy using socket API procedures, it will listen through a port number which can opened using command prompt as:
rudy:start(8080).
This command will open the port 8080 for rudy server to receive request from the client. (start function is defined in the “rudy.erl” file). “init(Port)” function is defined in to initialize (or open)

a listening socket. Further “handle(Listen)” function will listen to the socket, after the connection has been made with the client it will pass the connection to “request(Client)” method. Here is our http parser is provoked and then reply is sent back to the client.

d)Listening, Accepting and replying using gen_tcp library: Built in functions of gen_tcp library helped in implementation of listening to a coming socket, accepting the request from client.

gen_tcp:listen(Port, Opt)

gen_tcp:accept(Listen)

Further connection is established and server receives the request from client and then sends back the response and closes the socket, implementation is done using gen_tcp built in procedures given below:

gen_tcp: connect(Address, Port, Options)

*Connects to a server on TCP port **Port** on the host with IP address **Address.***

gen_tcp: recv(Client, 0)

This method receives a socket and length 0 marks that all available bytes are returned.

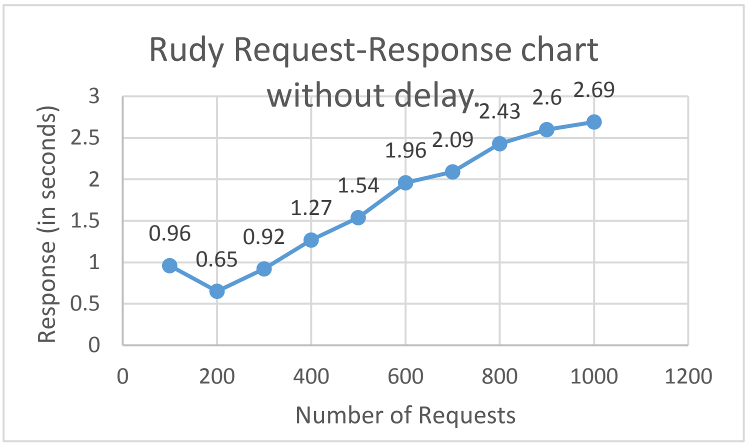
gen_tcp: send(Socket, Packet)

gen_tcp : close(Socket)

3. Evaluations:

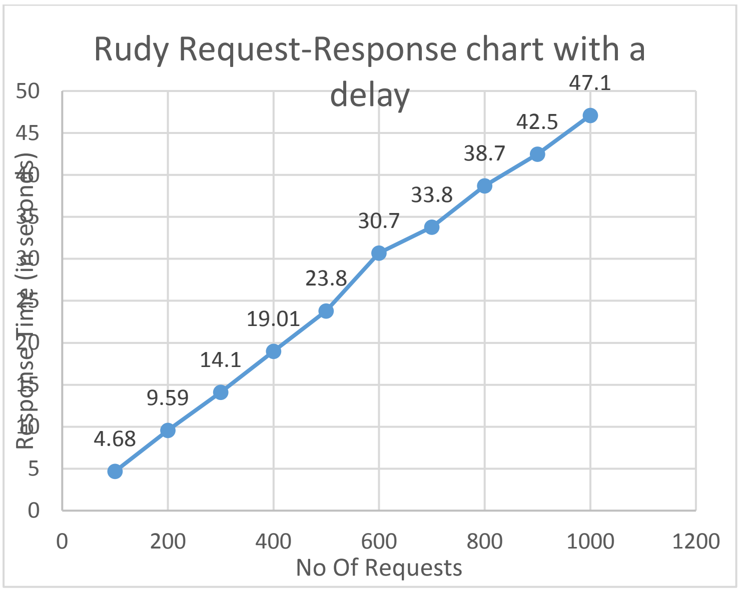
Using the “test.erl” benchmark program I have evaluated the performance of this webserver with delay and without delay, I have listed my findings below:

a) Without Delay:



No of Requests	Response Time (in seconds)
100	0.96
200	0.65
300	0.92
400	1.27
500	1.54
600	1.96
700	2.09
800	2.43
900	2.6
1000	2.69

b) With Delay 40ms



No of Requests	Response Time (in seconds)
100	4.68
200	9.59
300	14.1
400	19.01
500	23.8
600	30.7
700	33.8
800	38.7
900	42.5
1000	47.1

4. Conclusion

I have tested each scenario for the rudy server, graph showed almost linear curves with delay of 40ms compared to the without delay chart also when I run the benchmark program on different command prompt windows it shows the sequential results i.e. one after another showing that processing is sequential in the rudy server. Erlang's pattern matching and recursion functionalities are the core aspects used in the implementation of this server.