Report 2: Rudy homework report

Douglas Hammarstam

September 9, 2022

1 Introduction

In this project, a simple web server has been implemented.

In this seminar, the main topic covered was the HTTP-protocol and how to parse it using Erlang. Furthermore, the seminar covered how to use the gen_tcp package in order to receive network HTTP-requests and handle the requests by using above mentioned parsing of the HTTP request.

This is important in order to get an understanding of how to use Erlang in order to efficiently parse HTTP-requests (strings) using recursion.

It is also important in order to understand basic Erlang features like recursion, pattern-matching aswell as spawning processes in order to distribute an application on multiple threads.

2 Main problems and solutions

In order to build a small web server i Erlang we use the gen_tcp package which has an API for listening to incoming HTTP-requests on a given port.

```
gen-tcp:listen(Port, Opt)
```

The package also provides the possibility to accept incoming TCPconnections.

```
gen_tcp:accept(Listen)
```

Where "Listen" is port to listen on.

After that, we can handle each request by spawning a new process for each request. This makes the program more responsive to many requests at the same time, since we are able to handle requests concurrently.

```
spawn(fun() -> request(Client) end),
```

We also try limiting the amount of processes being created in order to create a upper bound on the amount of threads being created by the computer

In order to handle these requests. We need to do some parsing of the http-request.

The first problem in the assignment was to parse the incoming HTTP-request. This was done by dividing the problem into different sections; Firstly, parsing the method used in the request (GET). Then the URL is received by parsing each character after the space (32) character after the GET string until we reach another space character (32). Secondly, we get the HTTP-version (v10 or v11) by simply pattern matching the string to either of those values. Thirdly, we parse a line-break character followed by the headers, which we also recursively parse. Fourthly we parse the body where we assume that the rest of the request is made up of the body, which is not always the case in a real world scenario.

3 Evaluation

If needed, you may provide figures or tables with main results evaluating your proposals. For each seminar, we will provide you with some guidance on which kind of evaluation you should do.

And Figures 1 and **??** shows how to add a figure with some results. These figures have been created with gnuplot. There are tons of different kinds of plots that can be generated with gnuplot. Make sure to check out `http://gnuplot.info/demo/` and look at them so you can see what can you do with this program.

To obtain these figures you have to:

1. Create the data file from the experiments (look at file experiment.dat)

2. Create a gnuplot file to create a figure in eps format (look at files results1.plot and results2.plot). These files may be very complex. But for the results we want to show, these examples are enough. To create the eps figures, execute in terminal:

   ```
   $> gnuplot results1.plot
   ```

3. As pdflatex does not recognize eps files, you must convert them to pdf files. This is done by (it will generate a file results1.pdf):

   ```
   $> epstopdf results1.eps
   ```

4. That's it! Just include the figure as shown in this template and compile the latex as explained in the document "Introduction to LaTeX 2_ε".

If you want, you can also create a table of results as Table 1. If you look at the template code, you will see how to do a table in LaTeX.

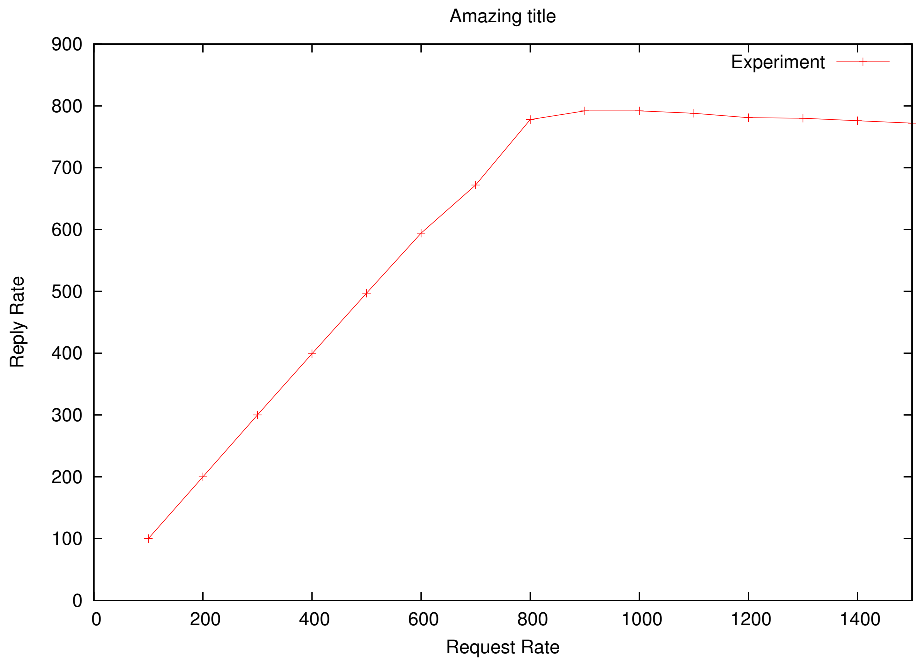

Figure 1: Some random results 1

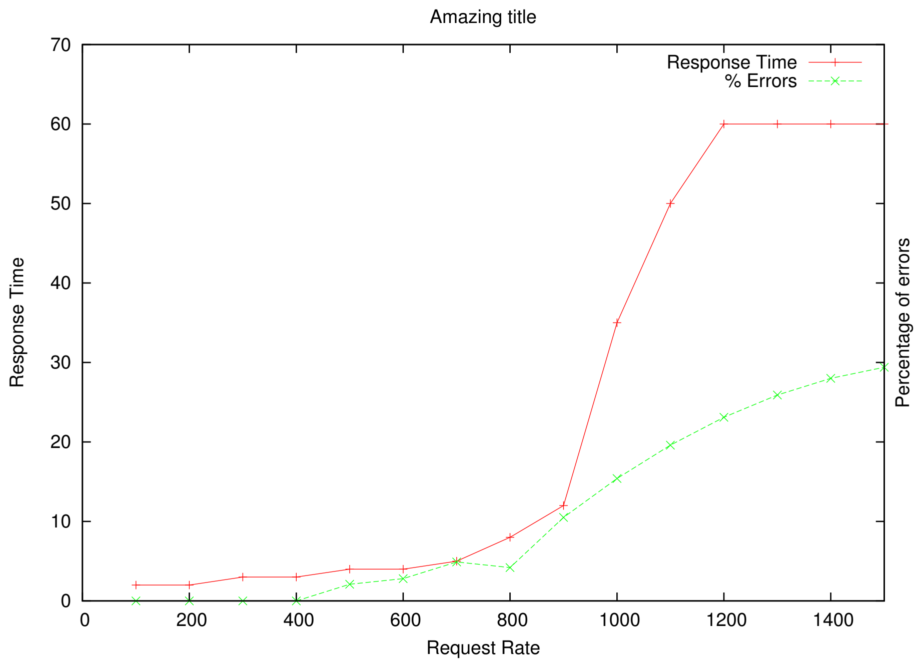

Figure 2: Some random results 2

First column	Second column	Third column
Case 1	1.1	1.2
Case 2	2.1	2.2
Case 3	3.1	3.2

Table 1: Some random results in a table

4 Conclusions

What have you learnt from the problem presented? Was it useful?