Report 1: Rudy - A Small Web Server

Michail Roussos, michailr@kth.se

September 10, 2023

1 Introduction

In this exercise we created a rudimentary server and performed some basic experiments to check the efficiency of the system when certain delays are added in the response and when a specific number of tests are executed. On top of the basic functionality, the file delivering functionality was implemented.

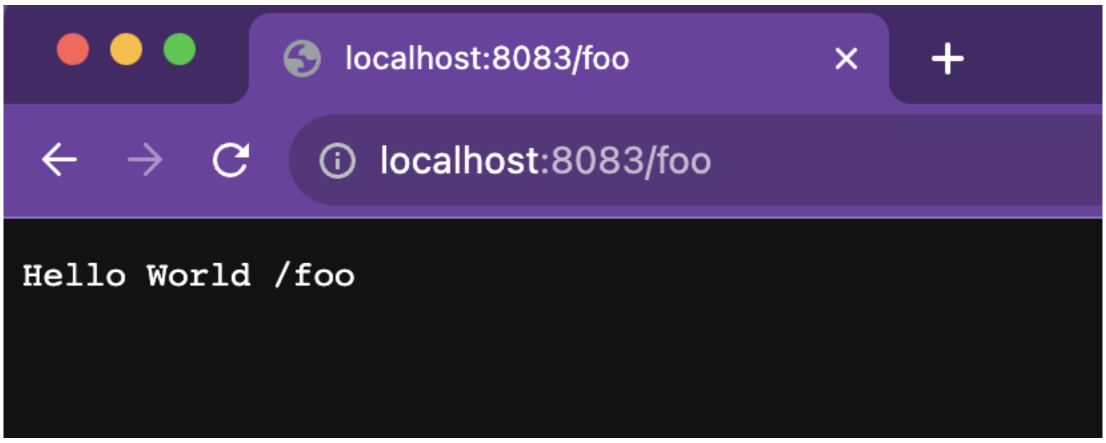
This exercise provided the opportunity to become familiar with servers/clients and the communication between them using Erlang! The communication between different kind of systems is what enables distributed systems.

2 Main problems and solutions

The task we were given, was to implement an HTTP server in Erlang. When starting the server we spawn a process that will open a TCP socket and wait for an incoming network connection. After the connection is completed the server waits for an incoming request and creates a response!

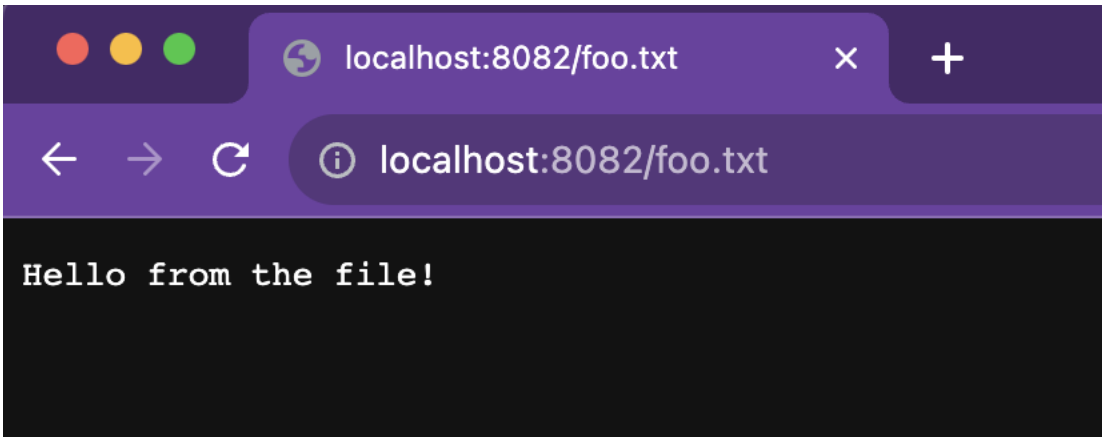
For the basic version of the server it only shows a String message. However, I also implemented the File Delivering functionality! For the file delivering functionality to work the files need to have a specific structure. The file needs to be in a folder 'files' which should be in the directory where the rest of the Erlang files are.

When the basic version is running and you navigate with a browser to the URL shown below you get a hello message and the URI.



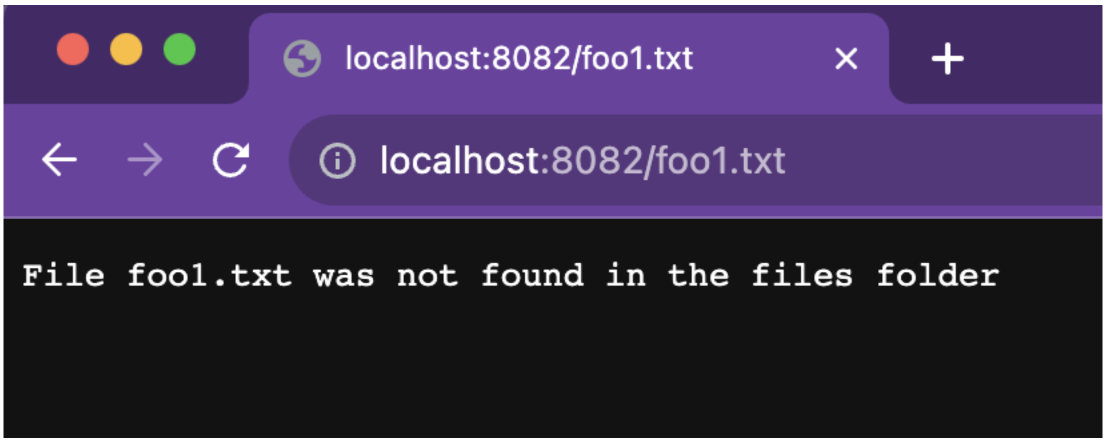
```
Hello World /foo
```

When the version that supports file delivery is running and you navigate with a browser to the URL shown below (where 'foo.txt' is a file inside the 'files' folder) you get the contents of the file.



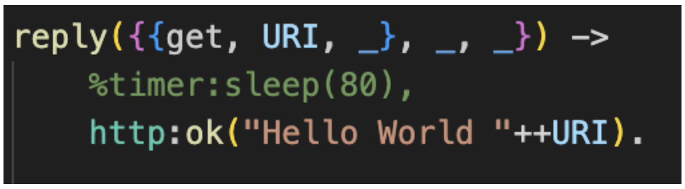
```
Hello from the file!
```

When the version that supports file delivery is running and you navigate with a browser to the URL shown below (where 'foo1.txt' is not a file inside the 'files' folder) you get the error message shown bellow.



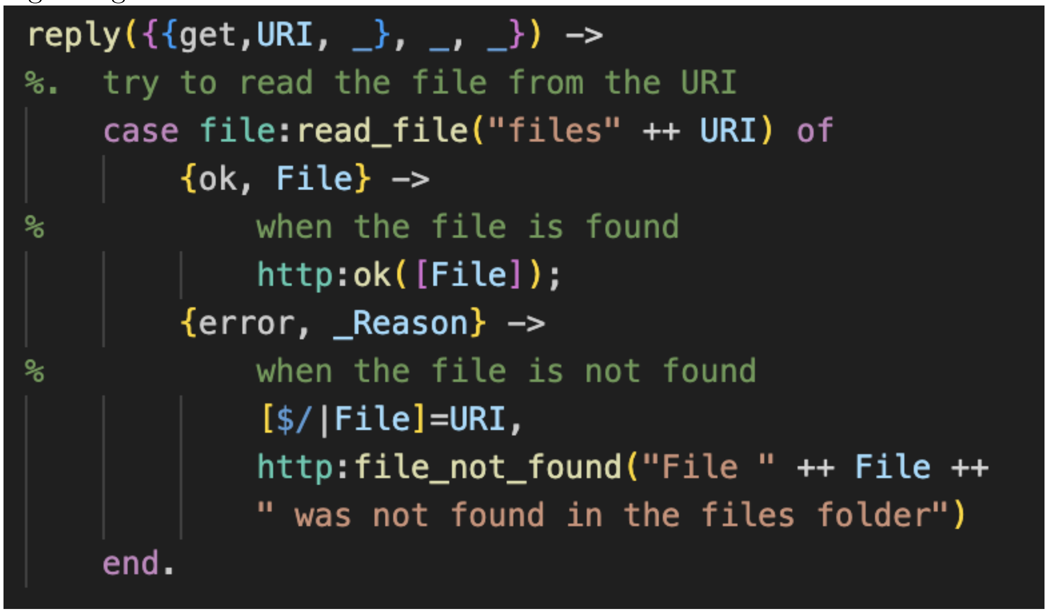
```
File foo1.txt was not found in the files folder
```

Afterwards I implemented the delivering files extra functionality. In order to achieve this I had to alter the following function rudy:reply/1. The initial version it this one:



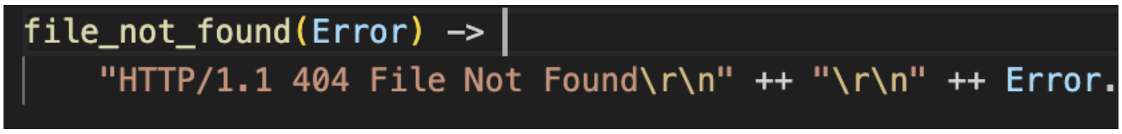
```erlang
reply({{get, URI, _}, _, _}) ->
    %timer:sleep(80),
    http:ok("Hello World "++URI).
```

In order for our server to be able to return files we need to make the following changes:



```erlang
reply({{get,URI, _}, _, _}) ->
%.  try to read the file from the URI
    case file:read_file("files" ++ URI) of
        {ok, File} ->
%               when the file is found
                http:ok([File]);
        {error, _Reason} ->
%               when the file is not found
                [$/|File]=URI,
                http:file_not_found("File " ++ File ++
                " was not found in the files folder")
    end.
```

I also created this function in order to produce an indicative error:



```
file_not_found(Error) ->
    "HTTP/1.1 404 File Not Found\r\n" ++ "\r\n" ++ Error.
```

3 Evaluation

After implementing the functionality some texts were executed to check how efficient the system is. I ran most of the scenarios with the basic version and then executed a few with the file delivering version as well.

The following tests where executed with the basic functionality: *(All the times are measured in microseconds)*

1. No Added Delay and 100 tests
2. No Added Delay and 200 tests
3. No Added Delay and 300 tests
4. 40 microseconds Delay and 100 tests
5. 40 microseconds Delay and 200 tests
6. 40 microseconds Delay and 300 tests
7. 80 microseconds Delay and 100 tests
8. 80 microseconds Delay and 200 tests
9. 80 microseconds Delay and 300 tests

Delay \Number of Tests	100 Tests	200 Tests	300 Tests
0 delay	43936	59781	96056
40 delay	4126210	8345108	16368806
80 delay	8183582	12525846	24575180

Table 1: Results from tests done with the basic version

I tested the following scenarios with the extra functionality:

1. No Added Delay and 100 tests
2. No Added Delay and 200 tests
3. No Added Delay and 300 tests

Delay \Number of Tests	100 Tests	200 Tests	300 Tests
0 delay	57093	84395	107370

Table 2: Results from tests done with the file delivering version and no added delay

4 Conclusions

In this exercise I became familiar with erlang and I had the opportunity to see how we can set up a server and a client using the HTTP and TCP stack.