SEWS (Zeus)

Simple Erlang Web Server

Process Oriented Programming (1DT038) spring 2011, final report for group 6, 2011-05-30.

Jonathan Boellke
Daniel Flodin
Alexander Fougner
Johannes Henriksson
Dennis Rosén
Per Wiberg

Abstract

SEWS was originally set out to become a new groundbreaking web server application. However, when we started SEWS we had no idea that we would be undertaking a fantastical journey throughout the intricate web of concurrency programming. During our programming sprints we managed to develop most of our original ideas, like downloading and uploading files, implementation of our own dynamic web scripting language, and a working cache for speed optimization. This has been an invaluable experience and we'll never forget the time we spent together with SEWS and each other. Unfortunately this also means we'll never forget the time we spent with EMACS - zing!

Now, read on if you dare. But watch out - you may not be able to stop reading!

Also, SEWS fights for the users!

Introduction

SEWS is your typical web server. This means that it is a program used for storing files on a server. SEWS sports the amazing features of downloading and uploading files, and you can even embed erlang code in special .esl files to get infinite functionality. SEWS is also incredibly fast (actually the fastest web server we have ever done!) - powered by our groundbreaking Cachezilla module it smashes any and all opposition. Cachezilla also implements a dynamic selection system for the caching algorithm used, providing ubiquitous flexibility. SEWS presents all this in a beautiful HTML environment using the latest in the HTML 1.0 protocol.

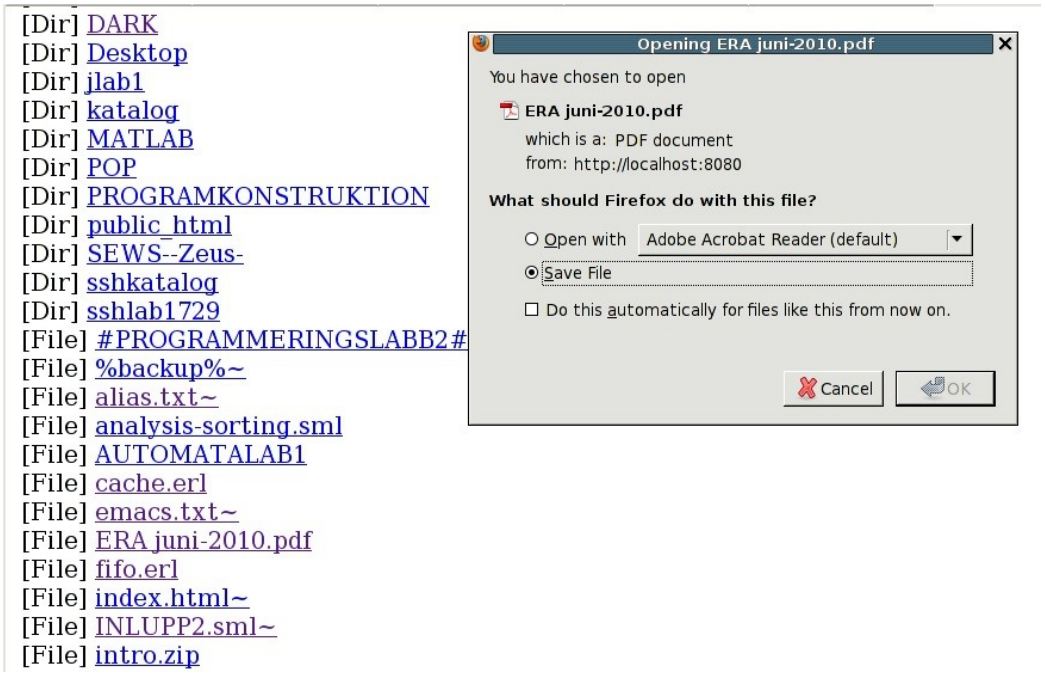
Why SEWS you ask? Well, today most users have a need for storing files in the “cloud”, thus allowing the user to access his/her files wherever he/she might find themselves in our universe. With this in mind we set out on our quest to conquer the cloud, and we did it all in the name of the user. Many a webserver has already been written in Erlang, but most of them have been implemented using the BIF from Erlang. We decided to make SEWS from scratch as it would be a great learning experience for all of us within the field of concurrent programming. Therefore SEWS is a one of a kind webserver written in Erlang.

SEWS - The Simple Erlang Web Server

Using SEWS the GUI is a web browser, which browser to use is up to the user as SEWS is compatible with the most common browsers. The GUI is simple and clean and gives the user all information needed about each file, be it a directory or file as shown in picture 1.

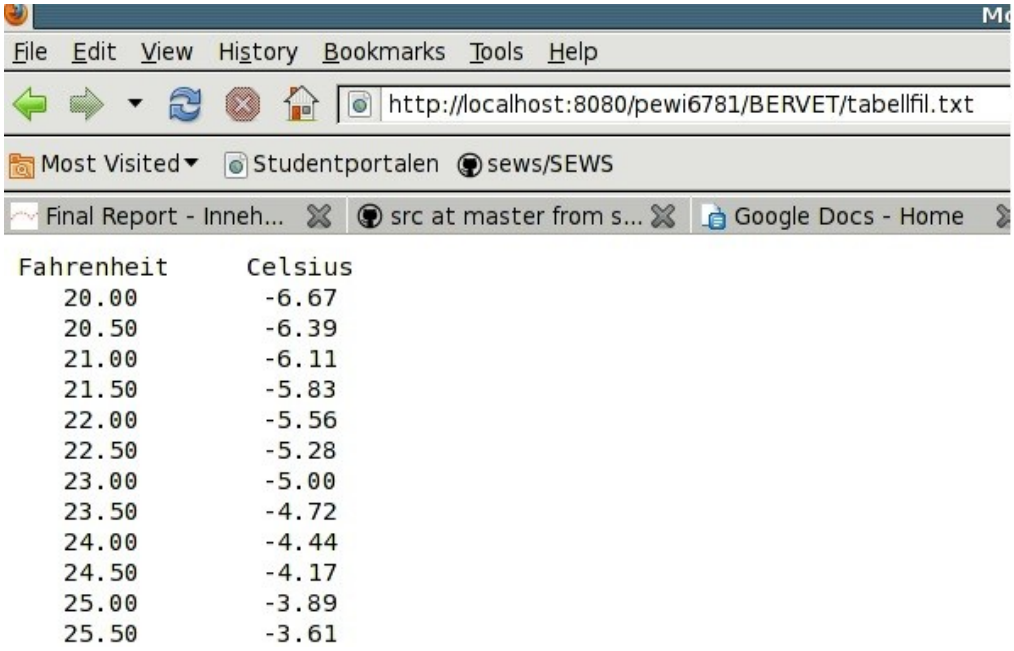
picture 1

As a user you are download files as you can see in picture 2. Uploading files is done by the host simply adding the html file upload.html in the folders where the host wishes to allow users to upload to.



picture 2

Choosing cache size is made an available option as well as choosing the preferred cache replacement algorithm, but only for the server host. In addition to this all unknown file types and .txt files will be shown directly in the browser, see picture 3.

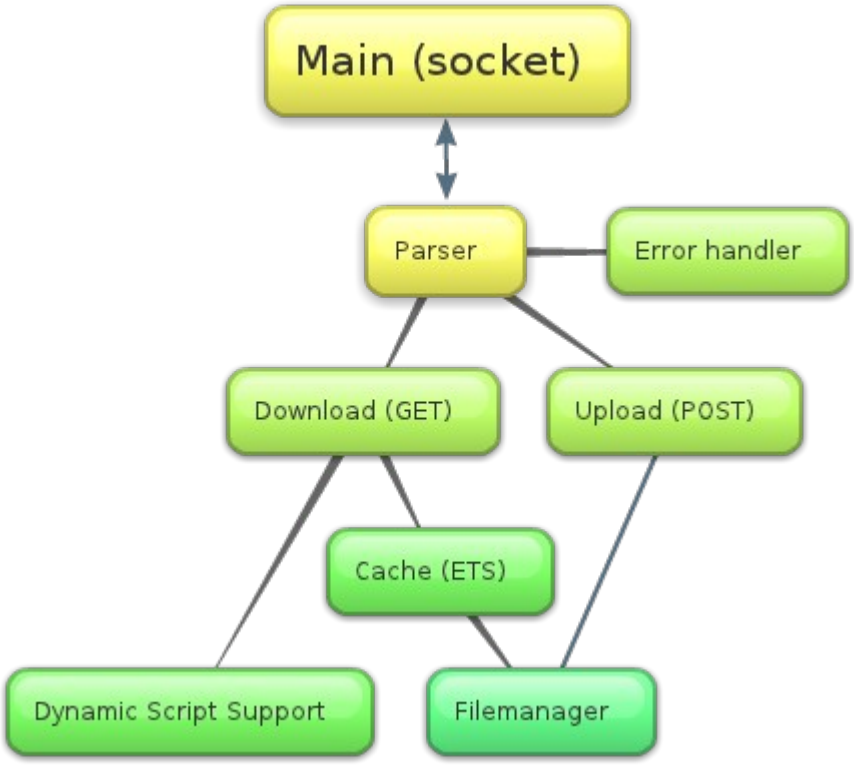


picture 3

System architecture

SEWS works in such a way that when a request is sent, it is picked up by the Main module.
In the Main module we have a loop for receiving socket packets. Main also contains an ETS table containing all files available (working as a cache). When a request is sent the Main module spawns a new process using the Parser module, see picture 4, which parses the received string into an acceptable format, and decides if the request should be sent to the GET or POST module. If an error occurs during parsing the Error handler is called.

GET and POST uses the parsed data to specify the request to send to the File Manager Module. Before the GET module calls the File manager it calls the Cache to see if the requested file has been called recently. If so the file is returned, otherwise the request will be passed on to the File Manager. When the File Manager receives a call it finds the specified file, returns a list of files in the folder or uploads the file and returns an updated file list. If the GET module has received any dynamic Erlang scripting it calls the Dynamic Script Support module which will execute the code and return the answer as a html page.



picture 4

Evaluating and testing

We ran some benchmark tests on SEWS and compared to the same tests when ran on Apache. For sending 1000000 requests while running 10 requests at any given time:

SEWS (ms)
50% 5
66% 5
75% 5
80% 6
90% 6
95% 6
98% 7
99% 7
100% 3016 (longest request)

APACHE (ms)
50% 3
66% 3
75% 3
80% 3
90% 3
95% 3
98% 3
99% 3
100% 3021 (longest request)

As we can see Apache serves 99% of all requests in a timely manner, 3 ms to be precise. SEWS on the other hand takes roughly double the time to serve 99% of its requests.

For sending 100000 requests while running 100 requests at any given time:
SEWS (ms)
50% 13
66% 14
75% 15
80% 15
90% 17
95% 18
98% 22
99% 215
100% 9043 (longest request)

APACHE (ms)
50% 31
66% 31
75% 31
80% 31
90% 32
95% 32
98% 32
99% 32
100% 47 (longest request)

In this case SEWS runs faster than Apache in 98% of the requests. This might be a result of SEWS being written in the comcurrent programming language Erlang compared to Apache which is written in C.

Related work

Yaws is an example of a webserver written in Erlang. The main difference between YAWS and SEWS is that YAWS uses the BIF from Erlang whereas SEWS was written from scratch by the geniuses of group 6. However SEWS has the option of using cache with any replacement algorithm the user might want! Thus [SEWS > YAWS] = true.

Installation instructions

Default options

Using the CLI, enter the SEWS folder and run the command

```
make
```

which compiles SEWS with default options:

Port: 8080
cache size: 50 elements
file size boundary in cache: 2 mb
replacement algorithm: LRU

Then type

```
erl -pa ebin
```

to start the erlang shell from the ebin folder and then write

```
main:start().
```

Then open your browser and enter the following adress to run SEWS from your browser:

```
http://localhost:8080
```

To change port and other options

open config.hrl file in the include folder and change the wanted options. Then run SEWS as if it was to be run with default options.

Conclusions

We have learnt how the architecture behind a webserver is built up and and what functions are needed to create a basic server. We met our goals of implementing GET (download), POST (upload) and a cache. After the second sprint we decided to implement dynamic Erlang scripting, thus allowing a request to a .esl-file containing Erlang code and executing said code.

Future work

Code optimization. Keep-alive. Update SEWS to HTTP 1.1 from HTTP 1.0. Toolbar to allow the user to enter scripts for SEWS. Extensive documentation of SEWS would be great. Process supervisor could be implemented **if** all code was rewritten. Erlangs BIF for process supervision can only be implemented with the BIF for gen_server, FSM and others.

Sources

https://pingpong.uu.se/courseId/6939/content.do?id=13562432 28/4-11

http://jmarshall.com/easy/http/ 28/4-11

http://en.wikipedia.org/wiki/KISS_principle 28/4-11

http://en.wikipedia.org/wiki/User_story 28/4-11

http://bubbl.us/ 28/4-11