# Assigments week 7

*Joe Armstrong*

February 7, 2014

## Contents

## 1 Introduction

These problems are for lectures F8 and F9, held in week 7 of 2014.

Additional material can be found in Chapters 11 to 14 of the course book *Programming Erlang, 2'nd edition*.

## 2 Problems

This week I've put the problems into three categories:

- **normal** solve these to get *Godkänd*.
- **advanced** solve these to get *Väl Godkänd*.
- **optional** solve these to get a deeper understanding of the problem. You don't have to hand in your solutions to these, but solving them, or attempting them will deepen your understanding of Erlang.

The sections and subsections in this paper are marked appropriately.

## 3 normal: Reading Configuration Data

Assume we have a file of configuration data `config.dat` such as the following:

```
{port,200}.
{hostname,"www.goole.com"}.
{interval,10}.
```

This contains Erlang keys and values.

Write a function `get_config(File, [{Key,Default}]` that finds the the value of various keys in the file. If the key is not present in the file, then the default value is taken.

For example:

```
> get_config("config.dat", [{port, 4000}, {timer,200}]).
[{port, 200}, {timer,200}]
```

The value of the function is a list of keys and value, where the value of the key come from an appropriate tuple in the configuration file, or from the value give in the argument to `get_config` if there is no value in the configuration file.

## 4 normal: Reading text files

Write a function `read_and_number_lines(File)` that reads a text file and breaks it into lines. It should then number the lines, returning a list of tuples, contain the line number and the line of text.

For example if the input file `abc` contained the following:

```
hello
this
is a line
with four lines
```

Then:

```
> read_and_number_lines("abc").
[{1,<<"hello\n">>},
 {2,<<"this\n">>},
 {3,<<"is a line\n">>},
 {2,<<"with four lines\n">>}]
```

## 5 normal: Listing files in a directory

Write a function `list_dir(Dir)` that returns a list of all the items in the directory `Dir`. The items returned in the list should be tuples of the form `{file,FileName,Size}` if the item is a file or `{dir,DirName}` if the item is a directory.

For example:

```
> list_dir("/home/joe/problems").
[{file,"readme", 200},
 {dir,"src"},
 {file,"problem.erl", 450},
 ..]
```

Hint: lookup the manual pages for the modules `filelib` and `file`.

## 6 normal: Reverse the order of the bytes in a file

Use `file:pread` and `file:pwrite` to do this. Is there and easier method?

## 7 advanced:Making a File Digest

A file digest is set of checksums that can can be used to check the validity of a file. To compute the digest we split the file into a number of fixed size blocks and compute a checksum for each of the blocks. If the file length is not be an exact multiple of the blocksize we aslo add a check sum for the last incomplete block in the file.

As an illustration we'll assume a file containing exactly 20 bytes containing the unsigned bytes [0,1,..19] and a block size of 6

The block contents are:

```
block 1: 0,1,2,3,4,5
block 2: 6,7,8,9,10,11
block 3: 12,13,14,15,16,17
block 4: 18,19
```

The digest of the file is made by computing the MD5 checksum of each of these blocks, and including information about the file size and the block size of the individual blocks. We can compute this and write the result to a file as follows:

```
Digest = {size,20,blocksize,6,checksums,
           [erlang:md5(<<0,1,2,3,4,5>>),
            erlang:md5(<<6,7,8,9,10,11>>),
            erlang:md5(<<12,13,14,15,16,17>>),
            erlang:md5(<<18,19>>)]},
file:write_file("digest", term_to_binary(Digest).
```

20 is the file size (computed with `file_lib:file_size`)
Write the following functions:

### 7.1 `compute_digest(File, Blocksize)`

This computes the digest of a file and stores it in a file called `File.digest`.

### 7.2 `check_digest(File)`

This assumes you have two files `File` and `File.digest`. Compute the digest of the data in File and check that is same as the digest in `File.digest`

## 8 normal:Sockets

Start by downloading `udp_test.erl` and `elib2_aes.erl` from `http://github.com/joearms/paradis`.

Modify `udp_test.erl` to make a simple "email like" server. Assume that port 2000 on your machine is a UDP email server.

If an Erlang message of the form `{email, From, Subject, Content}` arrives at port 2000 then store the message in some file (say `${HOME)/inbox`).

Note: the message should be serialized using `term_to_binary` and the variables in the tuple are assumed to be Erlang binaries.

Add a layer of encryption using the code in `elib2_aes.erl`.

## 9 optional: Make a secure file transfer program

If you've been doing the exercises you can now make a file digest, and should be able to setup a UDP or TCP client and server.

You can now make a secure file transfer program. First compute the digests for the files to be transferred. Break the file(s) to be transferred into a series of blocks, using the same blocksize as that used in the digests.

Transfer the blocks one at a time, checking the data as it arrives with the checksums in the digest. Use a blocksize of 1MB and TCP.

You should be able to interrupt the program at any time and restart it if it was interrupted.