DIT027 – Assignment 5

The exercises are supposed to be done individually. However, it is allowed to discuss the problems with other students and share ideas with other students. There are 6 parts in this assignment. Solving the first four of them is mandatory.

Don't forget to write comments in your code to make your solutions clearer!

There are three skeleton files: **assignment5.erl**, **assignment5_calc.erl** and **assignment5_registry.erl** that should be used in different parts of the assignment.

The submission deadline is **5 October 2016** at 23:55.

There will be no test suite provided.

In this assignment you are free to use any **library functions**, unless any specific restrictions are mentioned.

Overcomplicated solutions will not be accepted.

Make sure that you follow ALL the instructions closely!

GOOD LUCK!

Problem 1– pmap

Reimplement **pmap()** from assignment 4, so that it crashes when at least one of the function invocations it executes crashes, instead of returning some of the results. Furthermore, the function should cancel all remaining computations once one of them crashes before terminating with a crash. Please use Erlang links to solve this problem. Please implement your solution in the **assignment5.erl** skeleton file.

Problem 2 – Downtime

A. Your system consists of 10 physical nodes connected with a redundant network. The system is designed in such a way that all nodes must be running for it to provide service. Each node has a probability of failure of 1% in a given month, and it will take about 6h to fix a failure (or replace a node). Estimate the expected downtime per year for this system.

B. Your system consists of 10 physical nodes connected with a redundant network. The system is designed in such a way that 9 out of 10 nodes must be running for it to provide service. Each node has a probability of failure of 1% in a given month, and it will take about 6h to fix a failure (or replace a node). Estimate the expected downtime per year for this system.

Please write your answers as comments in **assignment5.erl**.

Problem 3 – Automatic clean up for the storage server

The storage server from Assignment 4 contains the **flush()** operation, which empties the storage for the calling process. However, when a process that uses the server crashes, it might not have emptied its storage prior to that. Indeed, we do not want to litter the code of the clients using the server with checking for errors, and calling **flush()** in exception handlers.

Instead, you should implement a feature in the storage server that performs flush any time a process that has stored anything on the server quits. The feature is safe, because no other process would be able to access the removed data anyway. The feature should be implemented using Erlang monitors.

Please submit a modified **assignment4_storage.erl** file (change its name to **assignment5_storage.erl**).

Problem 4 – Defective server

The calc server defined in the skeleton file **assignment5_calc.erl** can be run in different variants. You can execute a correct version of it by running an example function **assignment5_calc:jobs(no_faults)**. By running the same function with the argument **transient_faults** you will execute a defective version–its operations would sometimes malfunction. The server is run with a supervisor (running **sup()**), which will restart it each time it crashes.

When you run **assignment5_calc:jobs(transient_faults)**, it might hang. Explain what is the problem. Does the server process crash?

To fix the problem start by adding timeouts to the **add()** and **mul()** operations. When a timeout occurs, the operation should crash. Check how your change influences the execution of the example.

Now you should add code handling the problem. Do not add it in the server code, nor in the **client()** function. Instead, please modify the **jobs()** function, so that when a fault occurs, the current job (invocation of client) is repeated. The particular run of **client()** can be repeated at most 3 times. If it still fails after 3 tries, and error should be reported. Please use exceptions to solve this.

Problem 5– Registry (optional)

Implement your own registry that provides **register()** and **whereis()** operations, which work in the same way as the standard library functions of the same name. The registry you maintain should be separate from the standard registry, and be implemented using a server process. You should provide the **start()** and **start_link()** and **stop()** operations for starting and stopping the server process. You may use the standard registry internally for registering your registry server. Please use the provided **assignment5_registry.erl** skeleton file.

.

Problem 6 – OTP gen_server (optional)

Rewrite the solutions to the problems 3, 4 and 5 using the OTP gen_server behaviour.

To solve this problem you will need to create a number of modules instead of using the provided skeleton files. Please place them within a separate directory in your zip file.