LINFO1131 Practical Exercises
Lab 9: Erlang OPT

This week, we will explore Erlang Open Telecom Platform (OTP), a set of libraries that contain all the boilerplate involved in writing typical applications. We will explore two archetypal OTP behaviors (process roles): `gen_server` and `monitor`. You may want to take a look at the official Erlang documentation for the specifications of `gen_server` and `process`. For more details about the use of `gen_server`, the Learn you some Erlang blog is quite complete.

To avoid all the messy command line evaluation, you should strive as much as possible to use a single `main/0` function and run it outside of the `erl` command line, like this:

```
$ erl –noshell –s <your_module> main –s init stop
```

It calls your main function, then calls stop to halt the Erlang vm. You should first compile your code like this:

```
$ erl –compile <module 1> <module 2> ... <module n>
```

A first counter

As we need a very simple application to iterate quickly, implement a simple server that will act as a counter. Your `main` function should behave like this, giving the output *Value is 2*:

```
main() ->
C = counter:start(),
C ! inc,
C ! inc,
C ! {get_value, self()},
receive {value, V} -> io:format("Value is ~p~n", [V]) end.
```

To keep things clean, you may want to separate the `counter` module from your `test` module implementing the `main` function.

Time to use the `gen_server`

Now, start again in a fresh file because we will define the same server based on the tt gen_server behaviour. *Do not forget to take a look at the official documentation.*

1. You should implement at least the methods `init`, `handle_call` (for `get_value`) and `handle_cast` (for `inc`). Implement also a `start` method that starts a stand-alone counter server using `gen_server:start_link()`[1].
2. Again, implement a `main` method (possibly in a different file) that calls your server through the implemented API.
3. What are the difference in the invocation of `inc` and of `get_value` ?

[1]Why should you use `start_link` instead of `start`?

We would like to get a notification on the next incrementation of the counter. We need to implement a `watch` entry point and warn the caller when the counter value changes.

1. You can implement `watch` with either `handle_call` ir `handle_cast`. Which one should be preferred? Implement both versions and compare both implementations.

2. How would you modify the state to authorize multiple process to use the `watch` entry point simultaneously? That means, if multiple processes want to get notified when the counter changes. Implement these changes.

Monitors

As you have certainly observed by now, a `gen_server` crashes when it receives unexpected messages. Do you understand why it happens? We would like to add a monitor that starts one counter, and restarts it three times before failing.

1. Create a trivial monitor that starts one counter, and restarts it three times before failing.

2. Improve your counter to make it compatible with the monitor. This includes providing a proper `start_link` function.

3. Test that the server restarts after a crash. (The server should still be alive, but it's value should be zero).

4. Try to crash the monitor by crashing the counter as many times as needed.