Concurrent Programming in Erlang

In this exercise we'll run through an example of hot code upgrade, applied to the frequency server.

Use the comments on this step to share your approaches to this exercise, any difficulties you encounter or questions you may have, and your solutions.

The supporting file `frequency2.erl` is available (as a zip file) under 'Downloads' below.

Injecting new frequencies

In our frequency server example, the set of frequencies available is hard-wired in the code. It is required to add a new functionality to the server, to inject a list of frequencies into the server for future use (in addition to the ones already available).

You should build a new version of the `frequency.erl` module that:

- can receive messages of the form `{request, Pid, {inject, Freqs}}`, where `Freqs` is a list of numbers;
- can process this message by a function `inject/2` (defined in a similar way to the `allocate` and `deallocate` functions);
- provides a function `inject/1` which gives a functional interface for the client to initiate an injection of frequencies.

Testing the modification

Run your frequency server from the Erlang shell and check that it has the behaviour you would expect. This would always be a preliminary to upgrading in running system: check that your upgrade behaves properly before you make it live!

Supporting live upgrade

We'll work through the live upgrade of a running frequency server in a series of steps.

1. As it stands the frequency server code cannot be upgraded live. Modify the (old) code so that it is susceptible to upgrade.
2. Run the server from the Erlang shell, and call `allocate/1` repeatedly to allocate frequencies until there are none remaining.
3. Compile the modified frequency module that includes `inject` functionality, and call `code:load_file(frequency)` to ensure that it is loaded.
4. Call the function `code:soft_purge(frequency)` – does it do what you would expect?
5. Call `inject/1` with a suitable argument to inject a set of new frequencies in the running server, and then call `allocate/1` to see the effect of the upgrade.

Discussion

You might like to use the discussion attached to this page to talk about other ways that hot code loading could be used in the frequency server case study, and the precautions that you would need to take to ensure that these upgrades worked without breaking the system.
