CptS 483/580

Homework #5 – Concurrent Erlang first assignment

Assigned: 3/10/14

Due: 3/24/14 11:59:59PM. Turn in to the Concurrent Erlang Angel Dropbox

In this assignment you will create an Erlang module containing functions for "microbenchmarking" the performance of a couple of concurrent Erlang features.

Implement your code for module `ring` in file `ring.erl` and turn in a file by that name along with a `comments.pdf` file that reports on what you found. Put both files in a zip file named `ring.zip` and submit it on the class turnin page.

The module must export a function `ring(nProcesses, nMsgs)` that does all the work of creating a process ring, sending `nMsgs` around it, and reporting the total time taken. (Grad students implement a second function, `ring2`, as well. See below.) The program should work for values of `nProcesses` and `nMsgs` into the tens of thousands. A *process ring* consists of a number of processes with each one sending messages to its successor and the last one sending messages to the first one.

You may write additional functions and you may export additional functions – it's good practice to include some self-tests in a module that can be called from outside the module. Of course, provide a comment for each function describing what it does. (Erlang comments are started with a % character and extend to the end of the line.) **DO NOT spew debugging messages into the output: there should be only one line of output for each call to the ring function.**

The purpose of this homework is to give you familiarity writing functional concurrent code and explaining how it works. In your comments file describe how your code works – in particular explain the way that the processes are created and why you chose to do it the way that you did: there are several options, for example you can have each process create its successor, or you can have one process that creates all of the processes. You should not send any messages around the ring until the entire ring has been created. And you have to figure out a way to determine when the last message has be received so you know when to print the amount of time taken. Explain your choices in your comments file.

Also in the comments file, provide a table or tables that report the amount of cpu time taken for rings of size 1000, 2000, 4000, 8000 to process 0, 1000, 2000, 4000, 8000 messages. That is a 4x5 table containing 20 different times. Note that the time for 0 messages gives you a good estimate of how long it takes to simply create a given number of processes. *(A wise programmer will run all the required tests programatically rather than manually!)*

Note that there are at least three choices for how to send the messages: 1) the first node can send a message and wait for it to arrive before sending the next message OR 2) the first node can send all of the messages then wait for them all to arrive OR 3) the first node can non-deterministically send and receive messages so that sending and receiving are interleaved in some indeterminate fashion.

Undergrads: pick one of the strategies in the previous paragraph and implement it or identify a different strategy and implement that.

Grad students: pick two of the above strategies and implement both. Use as little duplication of code as possible. I don't want to see two entire copies of the assignment: use parameterization to avoid code duplication. You will however have two exported functions from your module instead of just one. Call the second one `ring2`.

How to time execution. This is a code fragment that you can include/modify in some other function.

```
statistics(runtime),
statistics(wall_clock),
% do some work here
{_,Time1} = statistics(runtime),
{_,Time2} = statistics(wall_clock),
io:format("The work took ~p cpu milliseconds and ~p
wall-clock milliseconds", [Time1, Time2]),
```

Refer to the documentation at http://www.erlang.org/doc/ for explanations of the primitives used in the code fragment above. `statistics` is a function described in the `erlang` module within the `erts` section under Basic at the left side of the page. `io` is a module and described in the `stdlib` section under Basic.